O3BNN-R: An Out-of-Order Architecture for High-Performance and Regularized BNN Inference

Tong Geng, Ang Li, Tianqi Wang, Chunshu Wu, Yanfei Li, Runbin Shi, Wei Wu, and Martin Herbordt

Abstract—Binarized Neural Networks (BNN), which significantly reduce computational complexity and memory demand, have shown potential in cost- and power-restricted domains, such as IoT and smart edge-devices, where reaching certain accuracy bars is sufficient and real-time is highly desired. In this article, we demonstrate that the highly-condensed BNN model can be shrunk significantly by dynamically pruning irregular redundant edges. Based on two new observations on BNN-specific properties, an out-of-order (OoO) architecture, O3BNN-R, which can curtail edge evaluation in cases where the binary output of a neuron can be determined early at runtime during inference, is proposed. Similar to instruction level parallelism (ILP), fine-grained, irregular, and runtime pruning opportunities are traditionally presumed to be difficult to exploit. To further enhance the pruning opportunities, we conduct an algorithm/architecture co-design approach where we augment the loss function during the training stage with specialized regularization terms favoring edge pruning. We evaluate our design on an embedded FPGA using networks that include VGG-16, AlexNet for ImageNet, and a VGG-like network for Cifar-10. Results show that O3BNN-R without regularization can prune, on average, 30 percent of the operations, without any accuracy loss, bringing 2.2× inference-speedup, and on average 34× energy-efficiency improvement over state-of-the-art BNN implementations on FPGA/GPU/CPU. With regularization at training, the performance is further improved, on average, by 15 percent.

Index Terms—Machine learning, BNN, high-performance computing, neural network pruning, out-of-order architecture

1 Introduction

Deep-Neural-Networks (DNNs) are in widespread use due to their ability to learn well enough to achieve high accuracy [7], [8], [9], [19], [20], [34]. However, for many high-volume, but cost- or power-restricted applications, accuracy is not an absolute requirement [18], [31]. Rather, reaching a certain well-defined level of accuracy is often sufficient, but with low cost and low-latency–or even real-time response–being highly desired. This is especially true for IoT and smart-edge devices [17], [25], [35], [38].

Because they satisfy these requirements, Binarized Neural Networks (BNNs) [28] have recently received much attention. BNNs use single bits to encode each neuron and parameter, thus significantly reducing computation complexity (from floating point or integer to Boolean) and memory demand (from bytes per datum to bits). This can potentially reduce the inference delay by orders-of-magnitude at the cost of a small loss in accuracy. A recent study by Bethge *et al.* [2] shows that well-designed BNN structures can achieve comparable and even superior accuracy (70 percent Top-1 accuracy for ImageNet) compared with state-of-the-art condensed full-precision networks such as MobileNet.

Having only two values per neuron, a BNN's network structure is significantly different from a conventional DNN's. These differences expose various new optimization opportunities. For example, Umuroglu, *et al.*, [33] show that the Batch-Normalization (BN) functions in most BNNs can be simplified to a threshold-based compare and thus avoid the floating point (FP) calculation. Fujii, *et al.*, [6] use *neuron pruning*, which eliminates neurons in the case where the sum of weights is lower than a pruning-threshold, and retrains the network for this adjustment. By doing so the number of neurons, and so the associated computation, is reduced. The accuracy, however, is compromised.

The work here is motivated by these previous studies together with the following two observations. First, in a BNN, a neuron's output is a Boolean whose value is determined by comparing the accumulation of all dot-products of the edges linked to this neuron with a fixed threshold that has been determined during the training phase. The idea is that we can immediately cease further computation of the dot-product and return (a) 1 as soon as the current accumulation becomes larger than the activation threshold (Fig. 1 A); or (b) 0[1] as soon as it is found that the current accumulation has no chance of

- *Tong Geng, Tianqi Wang, Chunshu Wu, and Martin Herbordt are with the Department of Electrical and Computer Engineering, Boston University, Boston, MA 02215. E-mail: tong.geng@pnnl.gov, {tianqi, happycwu, herbordt}@bu.edu.*
- *Ang Li is with the PCSD, Pacific Northwest National Laboratory, Richland, WA 99354. E-mail: ang.li@pnnl.gov.*
- *Yanfei Li is with the College of Information Science and Electronic Engineering, Hangzhou, Zhejiang 310027, China. E-mail: aoxue18@zju.edu.cn.*
- *Runbin Shi is with the Department of Electrical and Electronic Engineering, Hong Kong University, Hong Kong. E-mail: rbshi@eee.hku.hk.*
- *Wei Wu is with the Program Model Team, Los Alamos National Laboratory (LANL), Los Alamos, NM 87545. E-mail: wwu@lanl.gov.*

Manuscript received 3 Feb. 2020; revised 9 July 2020; accepted 17 July 2020. Date of publication 3 Aug. 2020; date of current version 17 Aug. 2020. (Corresponding author: Tong Geng.) Recommended for acceptance by M. Becchi. Digital Object Identifier no. 10.1109/TPDS.2020.3013637

1. −1 is encoded as 0 in XNOR Net [28].

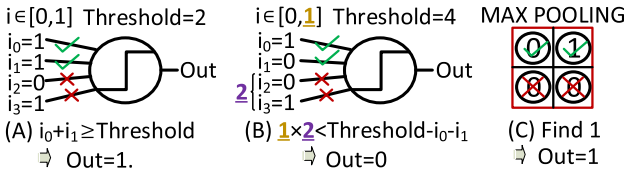


Fig. 1. Three types of pruning: (A) & (B) Threshold-based edge pruning; by accumulating the inputs (i_x) and comparing the accumulation result to a threshold, the value of a neuron (Out) is calculated (binary 1/0). (C) Pooling-based edge pruning. Out from pooling is binary (1/0).

reaching the threshold (Fig. 1 B). This cessation is analogous to breaking out of a loop as soon as the result is determined. We call this approach *threshold-based edge pruning* and refer to the two cases as *Conditions 1* and *2*.

A second observation is about pooling, in particular max-pooling, which for BNNs is the most widely. In the case where any one of the n×n inputs (typically 2×2) is 1, the pooling result must also be 1; thus we can avoid evaluating the remaining entries. For example, in Fig. 1 C the second entry is a 1 so we can prune the computation for the last two neurons. We refer to this approach as *pooling-based edge pruning*. Analogous methods work for min- and mean-pooling.

Although both observations are straightforward, the efficient harvesting of these pruning opportunities is challenging. This is because both are irregular, occasional, data-dependent, run-time, and strongly dependent on specific evaluation order. For threshold-based edge pruning, it is difficult to decide when the partial accumulation will surpass the threshold, or when we can assert that it will never reach the threshold. Pooling-based edge pruning is similarly difficult: it may eventually turn out that all entries are 0.

Exploiting these opportunities requires that the design be extremely flexible and dynamic. On the one hand, the control unit must frequently assess the current accumulation and be capable of immediately terminating the remaining execution of the neuron. This appears to require that the computation be sequential for the sake of pruning; yet, we still need parallelism to guarantee high performance. On the other hand, in case the evaluation of a neuron is terminated early, the execution gap needs to be filled instantly to avoid losing performance through pipeline bubbles. This combined challenge has been considered to be very difficult [6]. In this paper, we address these difficulties with an out-of-order edge pruning architecture: O3BNN-R.

To further enhance the potential gain, we propose an architecture/algorithm co-design approach during network training. We add two regularization terms to the loss function, for threshold-based and pooling-based pruning approaches, respectively. These two terms create more pruning opportunities without sacrificing accuracy by allowing the respective decisions to be made earlier: for threshold-based pruning, the regularization term moves the thresholds closer to either 0 or the maximum value; for pooling-based pruning, the term moves the 1 elements towards the upper-left corner of the pooling windows.

The main contributions of this paper are as follows:

- Two run-time approaches to edge pruning for BNN inference: threshold-based and pooling-based;
- A 2D-rotative out-of-order (OoO) design for dynamic workload scheduling and balancing;
- An architecture called *O3BNN-R* that implements efficient run-time BNN inference pruning; and
- Regularized training that enhances the pruning rate.

We evaluate the design on an FPGA platform using VGG-16 [30], AlexNet [22] for ImageNet, and a VGG-like network [4] for Cifar-10. Evaluations demonstrate that the out-of-order approach, without regularization in training, can prune 27, 19, and 42 percent of the operations for the three networks, respectively, without any accuracy loss. This brings at least 2.1×, 1.5×, and 1.7× speedups, respectively, and on average 47×, 23×, and 32× energy-efficiency improvements, respectively, over state-of-the-art FPGA/GPU/CPU BNN implementations. With training regularization, the performance of O3BNN-R is further improved, on average, by 15 percent and with only 0.5 percent accuracy loss.

The organization of this paper is as follows. In Section 2, we give BNN background and the motivate edge pruning. In Section 3, the edge pruning opportunities are introduced. In Section 4, an Out-of-Order BNN pruning design is proposed. In Section 5, the regularization augmentation is described. In Section 6, experimental results are presented and analyzed. In Section 7, related work is discussed. Section 8 provides a conclusion.

2 BNNs and Motivation for Pruning

2.1 Basic BNN Structure

BNNs evolved from conventional CNNs through Binarized Weight Networks (BWN) [4] with the observation that if the weights were binarized to 1 and −1, expensive FP multiplications could be replaced with additions and subtractions. It was next observed that if both weights and inputs were binarized, then even the 32-bit additions and subtractions could be demoted to logical bit operations; XNOR-Net was proposed and has become one of the most researched BNNs. In XNOR-Net, both the weights and the inputs of the convolutional and fully connected layers (except the first layer) are approximated with binary values, allowing efficient implementation of convolutional operations via exclusive-NOR (XNOR) and bit-counting [5], [28]. In this paper we use the terminology from XNOR-Net [28].

In their basic structure BNNs have four essential functions in each CONV/FC layer: XNOR, population count (POPCOUNT), Batch Normalization, and Binarization (BIN) (see Fig. 2 A). The weights, inputs, and outputs are binary so multiply-accumulate in traditional DNNs becomes XNOR and POPCOUNT in BNNs. The output of POPCOUNT is normalized in BN as this is compulsory for obtaining high accuracy with BNNs. BN incorporates full-precision FP operations, i.e., two FP MUL/DIVs and three FP ADD/SUBs

$$y_{i,j} = \left(\frac{x_{i,j} - \mathbb{E}[x_{*,j}]}{\sqrt{Var[x_{*,j}] + \epsilon}} \right) \cdot \gamma_j + \beta_j. \tag{1}$$

The normalized outputs from BN ($y_{i,j}$), which are FP, are binarized in BIN by comparing with 0

$$x^b = sign(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{otherwise} \end{cases}. \tag{2}$$

Here, BIN acts as the nonlinear activation function. Max pooling can be required. Traditionally, pooling is between

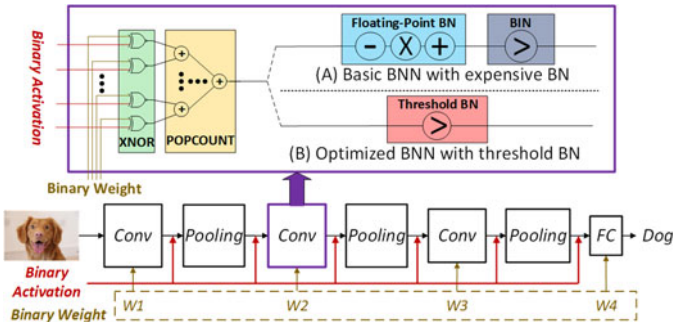


Fig. 2. A typical 3-CONV-1-FC BNN Network structure. It is similar to DNN, except that Activation acts as BIN, Multiplication acts as XNOR, Accumulation acts as POPCOUNT.

BN and BIN. It can be shown, however, that this is equivalent to placing pooling after BIN; thus the FP operations in pooling become bit-OR operations.

2.2 Motivation for Pruning

Researchers have observed various opportunities to further optimize the basic BNN structure. In FINN [3], [33] BN and BIN are merged. As shown in Fig. 2, the original FP-based BN function in Equation (1) and the BIN function in Equation (2) are integrated as a threshold

$$Threshold_{i,j,k} = \frac{\mathbb{E}_{*,j,k} + \mathbb{L}_{j,k}}{2} - \frac{\beta_{j,k} \cdot \sqrt{Var[x_{*,j,k}] + \epsilon}}{2 \cdot \gamma_{j,k}}, \tag{3}$$

where $\mathbb{L}$ is the length of the vector $K\times$K$\times$IC, K is the filter size, and IC is the number of input channels. Note that $\gamma_{j,k}$ and $\beta_{j,k}$ are learned in training and fixed in inference. In this way, the FP operations in BN now become a simple threshold. With a fixed threshold, our new observation is that we can prune certain computations in CONV/FC when a partial accumulation is already sufficient to obtain a result.

Another motivating study relates to neuron pruning [6]. In the FC layers, when the sum of weights for a neuron's linking edges are smaller than a threshold, this neuron is noted as inactive and is pruned. Fine-tuning is required and accuracy degrades. The authors also mention edge pruning, expecting that it can be more beneficial than neuron pruning, but do not pursue it due to the irregularity of the structures and the difficulty of the hardware implementation. Here we demonstrate that edge pruning is feasible and propose a dynamic out-of-order architecture that implements it with little hardware overhead, and with no (or managed) accuracy loss.

3 Pruning Opportunities

In this section, we first introduce the basic design of BNNs and then discuss the pruning possibilities. BCONV denotes bit convolution. The bit-fully-connected layers are treated as 1×1 BCONVs.

3.1 Basic BCONV Design

Fig. 3 shows pseudocode for a BNN with BCONV layers where K is the convolution filter size, N_{IC} is the number of input channels, N_{OC} is the number of output channels, $WIDTH$ and $HEIGHT$ are the width and height of the feature maps, and $LAYER$ is the number of layers.

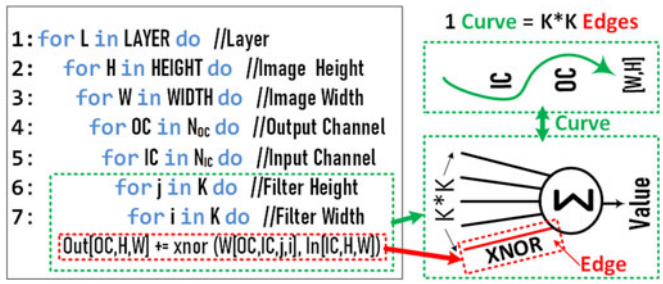


Fig. 3. Pseudo code of a traditional BCONV/BFC without pruning and the symbols of *edge* and *curve*.

There are 7 loops. Each iteration of Loop 7 processes an *edge*, i.e., XNOR + POPCOUNT, in the network graph (in red in Fig. 3). Each iteration of Loop 5 is called a *curve*; it processes $K \times K$ edges per input and output channel, i.e., a convolution window (in green in Fig. 3). The resulting value of a curve is the aggregation of its $K \times K$ edges. We annotate IC & OC along the curve to indicate the index of the curve in Loop 5 and Loop 4. We also annotate H & W at the front of the curve to indicate its index in Loop 2 and Loop 3. Therefore, a curve indexed by $[IC, OC, W, H]$ represents the workload of evaluating a convolution window of $K \times K$ neurons for input channel IC and output channel OC at location $[W, H]$ of the input feature map. The complete calculation of each output channel requires the accumulation of N_{OC} curves (Loop 4); the entire BCONV layer requires $N_{IC} \times N_{OC} \times HEIGHT \times WIDTH$ curves. In this design *a curve is the basic granularity for edge pruning*. Existing work [6], [27], [37] generally exploits parallelism in the loop-nest through:

- *Loops 6-7*: Parallel execution for $K \times K$ edges in a curve.
- *Loop 5*: Parallel evaluation of different input channels IC for the same output channel OC.
- *Loop 4*: Parallel processing different output channels OC.
- *Loops 2-3*: Usually not parallelized to ensure data reuse across neighboring $[H, W]$ (i.e., neighboring CONV windows overlap).
- *Loop 1*: Usually not parallelized since a hardware implementation to exploit model parallelism may suffer from layer-wise workload imbalance and excessive storage demand for intermediate results.

3.2 Threshold-Based Edge Pruning

Fig. 4 A illustrates threshold-based BN for each output channel (OC in Loop 4): (1) calculate and accumulate N_{IC} curves for this output channel, i.e., Loop 5; and (2) binarize via threshold comparison. Since the threshold is fixed in inference and the output is a binary value, we do not necessarily need to evaluate and accumulate all the curves before making a comparison. In other words, if the partial results are already sufficient to imply the output bit, we can avoid the evaluation and accumulation of the remaining curves. In the following, we use **ACC_Cur** to denote the accumulated partial curves, **ACC** for the accumulation results for ACC_Cur curves, and **T** for the threshold.

Condition 1. *$ACC > T$ implies that the remaining $(N_{IC} - ACC\_Cur)$ curves can be pruned.* As both input features and

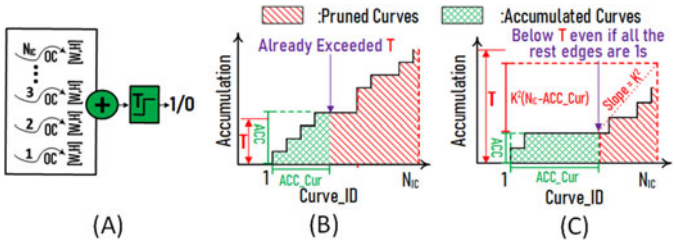


Fig. 4. (A): illustration of the evaluation process of an output neuron using threshold-based BN function; (B)&(C): Conditions of threshold-based edge pruning.

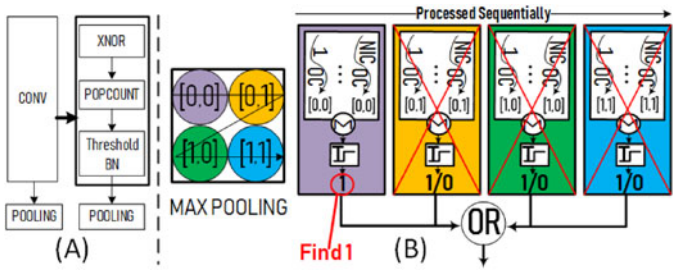


Fig. 5. (A): BNN structure used in this work: threshold-based BN followed by POOLING; (B): The condition of pooling-based edge pruning.

weights in BNNs are binary (1/0), the curve's value is always non-negative and accumulation never decreases ACC. Consequently, whenever ACC exceeds T, the binarization result is 1 and will never flip to 0 during the remaining accumulation. Therefore $(N_{IC} - ACC\_Cur)$ curves can be pruned, as shown in Fig. 4 B.

Condition 2. *$ACC < T - K^2 \times (N_{IC} - ACC\_Cur)$ implies that $(N_{IC} - ACC\_Cur)$ curves can be pruned.* Conversely, as the maximum value of each curve is K^2 (all XNOR results are 1), with ACC_Cur input channels already accumulated in ACC, for the rest evaluation of $(N_{IC} - ACC\_Cur)$ curves, the maximum possible contribution is $K^2 \times (N_{IC} - ACC\_Cur)$. Therefore, if $ACC + K \times K \times (N_{IC} - ACC\_Cur)$ is less than T, then ACC will never reach T; we can safely prune the remaining $(N_{IC} - ACC\_Cur)$ curves and output 0, as shown in Fig. 4 C.

Implementation Challenges. (1) To prune within Loop 5, it must execute sequentially; this may inhibit parallelism that can otherwise be exploited. (2) Due to pruning, the latency for each iteration in Loop 4 can differ substantially, which may lead to workload imbalance. (3) Dynamic, asynchronous, and data-dependent slacks must be filled immediately; otherwise pruning will not result in any performance benefit. (4) The hardware overhead for verifying pruning conditions, ceasing the present execution, and stealing new jobs for workload balancing must be limited.

3.3 Pooling-Based Edge Pruning

Given a threshold-based BN design, we now consider the pooling function (Fig. 5 A). Fig. 5 B shows how the four entries of a 2×2 pooling window are sub-sampled after a convolution. As the entries are binary, the *max* operation is equivalent to a bitwise-OR among the four entries. Therefore, once an entry is identified as 1 (e.g., the first entry in Fig. 5 B), the pooling result must be 1 and we can safely prune the evaluation of the remaining entries. For example, the convolution of three entries in Fig. 5 B is pruned.

Implementation Challenges. (1) To prune the pooling entries, the computation of these entries must be processed sequentially, limiting parallelism. (2) Pruning may lead to workload imbalance. (3) The dynamic and data-dependent slacks due to pruning must be leveraged effectively. (4) Extra delay and hardware overhead must be limited.

4 Out-of-Order BNN Pruning Design

A critical question to be addressed is the conflict between the need for sequential execution to facilitate pruning and parallel execution to obtain performance. In this section we first present a trade-off strategy and a method to compensate for compromised parallelism. We then show how to achieve workload balance via rotative workload scheduling. Finally, we discuss the O3BNN-R hardware implementation.

4.1 Parallelization Strategy

To achieve threshold and pooling edge pruning, Loop 5 must be executed sequentially and Loop 4 partially sequentially. To compensate for this reduced parallelism, we exploit the inter-layer parallelism (i.e., model parallelism) from Loop 1. Note that data reuse in Loops 2 and 3 is still critical for performance. We resolve layer-wise workload imbalance by allocating computation resources proportional to the per-layer workload. For large storage demand, we adopt a *layer-fusion* technique, as referred to in [1]. Overall, parallelism from K (Loops 6-7), OC (Loop 4), and L (Loop 1) are exploited for parallel execution. The pooling pruning at different output channels is applied in parallel. In the case that more parallelism is needed, Loops 2 and 3 can be unrolled and processed partially in parallel. By doing so, pooling pruning at the same output channel can also be conducted in parallel.

4.2 Rotative Workload Scheduling

For clarity and without loss of generality, let us first assume that 4 processing elements (PEs) process a BNN layer with 8 input and 8 output channels (IC=8, OC=8). We present three approaches to show the evolution of the design: *in-order*, *1D Rotative OoO*, and *2D Rotative OoO*.

In-Order Scheduling. All PEs work in lock-step. Whenever ACC (accumulation of curves) at one PE triggers one of the threshold pruning conditions, this PE aborts and remains idle (Fig. 6 A). The simple in-order design has two advantages. The first is low storage demand. Since the N_{OC} output channels can be statically partitioned among PEs (e.g., PE2 in Fig. 6 A always processes OC-2 and OC-6), the weights for convolution can be distributively conserved, saving memory space. The second advantage is simple data feeding logic under fixed curve mapping. The drawback is that it does not benefit significantly from pruning (except when all PEs are pruning, which is rare), while wasting computation resources and creating pipeline bubbles.

1D Rotative OoO. In the basic OoO design, a new curve immediately fetches and fills the gap from pruning. For example, in PE2, curves with OC=6 (in dark blue) issue after curves with OC=2 (in green) are pruned in cycle-3. To efficiently address the target curve location (using IC and OC) in the input feature map, which is stored sequentially in (possibly very large) memory, we propose a vertical rotative

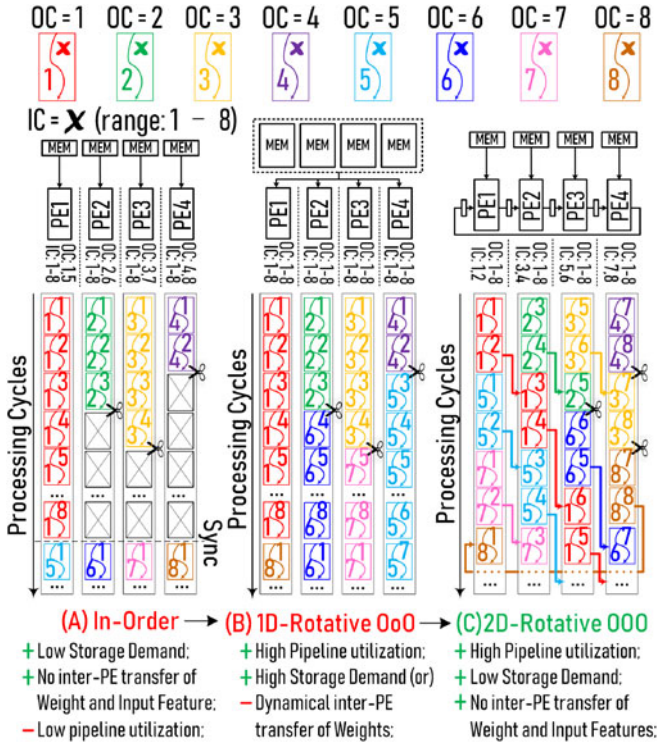


Fig. 6. Three methods of workload scheduling as described in the text.

design. We use a 1-8 counter for IC while dynamically controlling the value of OC for OoO. As shown in Fig. 6 B, PEs execute curves consecutively with IC rotating through 1 to 8. When pruning occurs, OC is updated following numerical order, i.e., new curves that have the next unprocessed OC ID are assigned. We refer to the group of curves with the same OC, but different IC, as a *curve-group*.

This design executes OoO to exploit pruning opportunities without introducing any pipeline bubbles. A major shortcoming, however, is storage cost: since it is not known in advance which OC will be fetched for a particular PE (pruning is data dependent), every PE can potentially process any OC with any IC. Therefore, every PE must retain a local copy of the entire set of weights. If the weights were shared globally, a very clever data-feeding circuit would have to be designed to issue the required weights to corresponding PEs on the correct cycle. This would introduce delay as well as area and power overhead. Also, neither approach is scalable to a large number of PEs.

2D Rotative OoO. To resolve the memory issue, we propose 2D rotative OoO. The idea is to distribute the weights among PEs using a time-sharing approach. Specifically, rather than partitioning curves among PEs along OCs (as with the in-order design), we partition along ICs. In other words, each PE statically handles a portion of ICs. For example, in Fig. 6 C PE3 only processes IC-5 and IC-6. Consequently, weights can also be statically partitioned along IC and distributively reside in the PE's local memory. For the horizontal rotation, PEs are connected to their right neighbors forming a unidirectional loop among PEs (horizontally in Fig. 6 C). When a PE finishes its portion of ICs, it forwards the unfinished curve-group to the side buffer of its right neighbor. To fetch a curve each cycle for execution, a PE first checks its left-side buffer and continues the unfinished curve-group in case presented; otherwise, it fetches a new curve-group (the current curve-group is either pruned or completed) and starts execution.

To summarize: by simultaneously rotating the curve groups along the vertical dimension, we dynamically dispatch them with desired pruning capability and with low memory addressing cost. This is done while distributively sharing weights among PEs in a time-sharing manner by rotating along the horizontal dimension. Given these advantages, the 2D rotative design is used for the O3BNN-R hardware implementation.

4.3 O3BNN-R Architecture

The O3BNN-R architecture is shown in Fig. 7. To achieve workload balancing, the PEs and other hardware resources are allocated roughly proportionally to workload per layer (shown in Fig. 7 A). In this way, layers linked in a daisy chain can cooperate effectively in a deeply pipelined manner, exploiting inter-layer parallelism from Loop 1. Each PE contains three major modules: *PE array* for workload execution, *Scoreboard* for tracking curve execution status and ensuring in-order commitment, and *data feeding system* (DFS) for buffering and feeding input data to the PE array.

4.3.1 Processing Element Array

Figs. 7 C&7 D show the detailed architecture of the PE array. To implement horizontal rotation, PEs are linked via a unidirectional circular communication network with two channels: one for forwarding the unfinished curve-groups (red-line) and one for conveying the current ACC values. Inside a PE are three buffers. (1) The buffer at the bottom-left is used to store and reuse input feature at a particular $[H, W]$ (i.e., reuse input feature data across OCs) with each curve per time slot. The 2-to-1 multiplexer linked to this buffer is used to select between reused input features for the next OC (i.e., curve-group) and buffering a new input feature from the next image pixel $[H, W]$. The FIFO loop-back implements the vertical rotation by repeatedly reusing the buffered data for different OCs. (2) The middle buffer is for distributively storing weights with each PE holding $N_{IC}/PEs * N_{OC}$ curve entries. The input feature and weights for a curve are XNORed and accumulated in parallel (into ACC). (3) The upper-right buffer is for pending data for inter-PE communication. The 3-to-1 multiplexer selects from: (a) 0-input for a new curve-group; (b) self input for accumulation within its curve-group portion; (c) neighbor input to start its portion of a curve-group following its left neighbor. The 2-to-1 multiplexer chooses between processing its curve-group portion and conveying the curve-group to its right neighbor PE.

4.3.2 Scoreboard

Analogous to reservation stations in Tomasulo's algorithm [15], [32] for exploiting instruction level parallelism (ILP) in an OoO CPU, the Scoreboard here tracks curve-group execution status and enforces commitment of curve-groups in the correct order. As shown in Fig. 7 E, each entry tracks a curve-group and has three basic fields: OC for curve-group ID, status for control, and the 1-bit output for this OC. Each column with N_{OC} entries in the same color tracks N_{OC} curve-groups for the pixels with the same [W,H] at all (N_{OC}) output

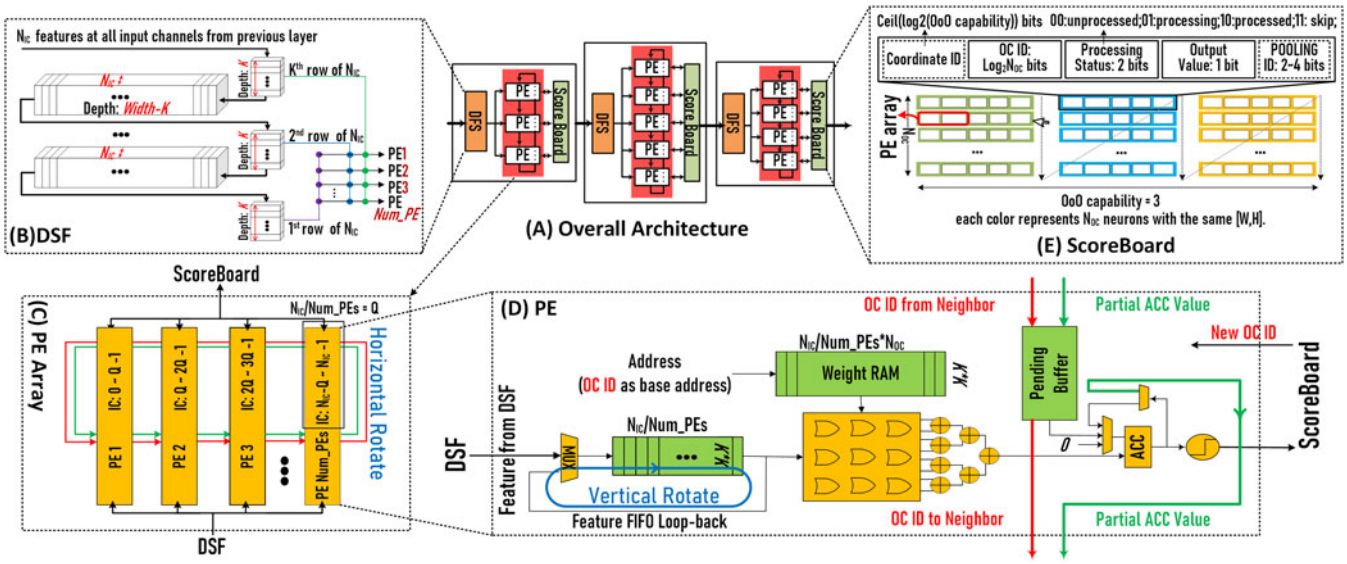


Fig. 7. (A): Overall architecture of O3BNN-R. (B) DFS architecture. (C) PE Array with illustration of horizontal rotate. (D) Architecture of each PE slice. (E) Scoreboard of which the OoO capability is 3.

channels. In our design, the 1-bit outputs of entries in the same column are committed to the succeeding layer simultaneously. Thus the number of entries in the Scoreboard is an integral multiple of N_{OC}.

We define the OoO capability of an O3BNN-R design as the number of column entries in its Scoreboard. The coordinate field is for tracking curve-groups with multiple [W,H]s and used when a multi-column Scoreboard is needed for stronger OoO capability (discussed later). The pooling field is used when pooling is required after the convolution, where curve-groups belonging to the same pooling window share the same Scoreboard entry. When performing convolution for the elements covered by the same pooling window, in case an element returns 1, the status field is marked as *skip* and the output field is set to 1. With skip status, the remaining elements sharing the same Scoreboard entry will not be issued to the PE array, i.e., they will be pruned.

4.3.3 Data Feeding System

The DFS is designed to store the input feature tensor that can be processed in the coming execution cycles and forward $K \times K \times N_{IC}$ features to a PE array in each cycle. For efficiency, a simple segmented line-buffer design is proposed (Fig. 7 B). The entire input feature map flows along each segment of the line-buffer with the K vertical segments extracting the required rows covered by the $K \times K$ convolution window and, when a new curve-group is requested, feeding them into the PE array.

4.4 Design Extensions

In this section, we sketch extensions to O3BNN-R. First, we describe adding a relaxing factor to the threshold and how this affects accuracy and performance. We then analyze how applying different OoO capabilities affects performance.

4.4.1 Relaxing the Threshold

None of the pruning designs described so far lose accuracy. But if accuracy can be compromised slightly, then more benefits can be gained. The idea here is to augment the threshold with a relaxing factor $\delta \in [0, 1]$.

For Condition 1, we relax T to a lower threshold, $\delta \times T$, so that it is triggered earlier and the calculation of a neuron stops before the accumulation result surpasses T. By doing so more operations are pruned, but accuracy may decrease: we now assume all curve-groups which surpass $\delta \times T$ eventually surpass T, which may not happen. For Condition 2, we relax T to a higher threshold, $(1 + (1 - \delta)) \times T$. Similar to Condition 1, pruning rates are increased while accuracy may decrease. This trade-off between accuracy and pruning is discussed in Section 6.1

4.4.2 OoO Capability

Because of in-order commitment, when the Scoreboard has only one column of entries it can only track N_{OC} curve-groups at the same time for the pixels with the same [W,H]. New curve-groups with new coordinates cannot be issued before the present curve-groups are completely evaluated, which may limit OoO capability. If more hardware resources are available for the Scoreboard, we can track multiple N_{OC} curve-groups at the same time (each per-column as shown in Fig. 7 E), by assigning different coordinates to different columns of entries. This is a trade-off between hardware consumption and OoO capability: a larger Scoreboard provides higher OoO capability.

Fig. 8 shows how different OoO capabilities affect the execution of BNN inference. Each block refers to the execution of a curve whose OC and IC IDs are indicated by the head of its row and column. To illustrate, we use a BNN layer with 8 ICs and 8 OCs. Each cluster of blocks (8×8 blocks in this figure) represents all curves that need to be evaluated to completely calculate features with the same [W,H]. The number on each block shows the iteration during which the curve is calculated. Different block colors represent execution at different PEs.

In Fig. 8 A, the Scoreboard only has one column of entries, i.e., OoO capability = 1. After 8 iterations, there are no pending curve-groups with [0,1] in the Scoreboard waiting to be forwarded to PEs. PE1 becomes ideal: it can start to work on

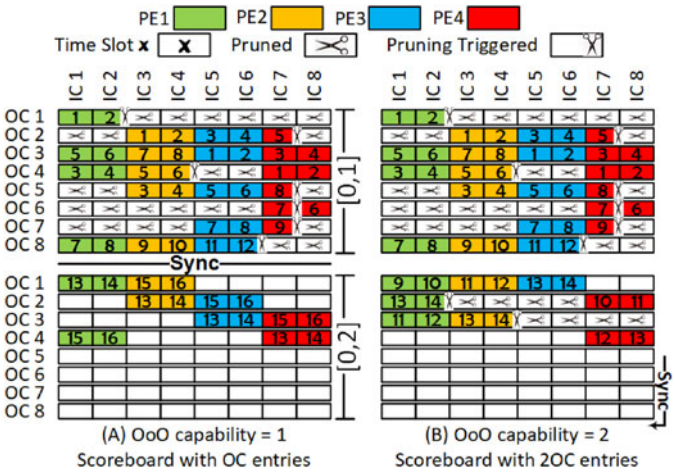


Fig. 8. Workload execution of O3BNN-R with different OoO capabilities. Four PEs are shown.

the curves of [0,2] at the 9th iteration. However, as the Scoreboard with OoO capability = 1 can only track N_{OC} curve-groups with the same [W,H] at the same time, the next N_{OC} curve-groups cannot be forwarded to PEs until the current ones are completely done and flushed from the Scoreboard. Therefore, synchronization is required between each cluster of blocks. In this example, synchronization takes 4 iterations out of 12 iterations of execution. At iteration 13, the calculation of curve-groups with [0,2] starts.

Fig. 8 B illustrates the execution of curve groups of [0,1] and [0,2] with a scoreboard having OoO capability = 2. After iteration 8, the second column of entries in the scoreboard starts to track the status of curve-group [0,2] immediately and without synchronization. As shown in Fig. 8 B, at iteration 9, idle PEs start to calculate the curve-groups of [0,2], 4 iterations earlier than (A). For O3BNN-R with OoO capability = 2, synchronization is required in the following scenario: when all curve-groups of [0,2] are already distributed to PEs by the Scoreboard and there are still curve-groups of [0,1] being processed in PE array. To avoid this synchronization, a Scoreboard with OoO capability = 3 is needed. The third column can then be used to track the curve-groups of [0,3]. Overall, the higher OoO capability the lower the probability that synchronization is needed. Based on our experiments, a 2-column scoreboard provides an optimal trade-off of performance and hardware consumption. This is discussed further in Section 6.2.

5 Regularized Training

So far we have focused on BNN inference, in particular, we have assumed that the weights for BNN inference are fixed. In this section we propose a training/inference co-design approach that enhances the utility of both threshold-based and pooling-based edge pruning.

5.1 Regularization for Threshold-Based Pruning

Threshold-based edge pruning is triggered under two conditions: (1) the accumulation has already surpassed the threshold; (2) the accumulation cannot reach the threshold even if all remaining partial results are 1 s. It follows that there is a higher chance of pruning when the accumulation

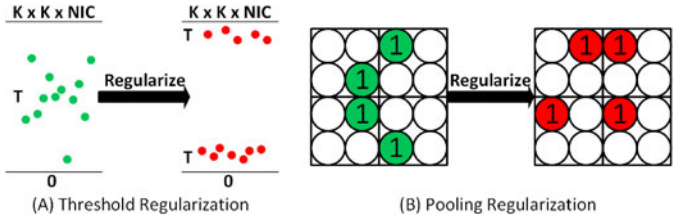


Fig. 9. Regularization during training.

stays away from the threshold, either larger (approaching max bound $K \times K \times N_{IC}$) or smaller (approaching min bound 0), as shown in Fig. 9 A. This can be achieved by adding a regularization term to the loss function during training. This term increases the loss value when the absolute difference between output and threshold diminishes, and decreases the loss value when the difference expands. As the SGD-based optimizer iteratively minimizes the loss function during training, we can obtain a BNN network model having a higher chance of early threshold-based pruning while suffering little accuracy loss. This is possible because of the redundancy in network parameters and input images, and because of the robustness of SGD optimization.

The threshold regularization term is defined as

$$R_{\text{Threshold}} = -\alpha \sum_{i=0}^{L\times H\times W\times N_{OC}} (y_i - \text{Threshold}_i)^2, \tag{4}$$

where α is a scalar weighting factor which can be adjusted to find the optimal point in the trade-off between pruning benefit and accuracy loss.

5.2 Regularization for Pooling-Based Pruning

As discussed in Section 4.1, the features covered by the same pooling window are evaluated sequentially. Therefore, in the case that one of these features is determined to be 1, all operations for evaluating the remaining features are pruned and the pooling output is set as 1. The evaluation of features covered by a pooling window follows a particular order, i.e., from top-left to bottom-right. For example, the order for a 2×2 pooling window is *top-left→top-right→bottom-left→bottom-right*. If during training we encourage the first features in the evaluation chain to be 1, the pooling-based pruning will be triggered earlier and pruning opportunities will increase (as shown in Fig. 9 B). This is achieved by adding another regularization term to the loss function.

We can encourage the 1 values to move forward in the evaluation chain by giving differently weighted rewards to different features covered by a pooling window. For example, for a 2×2 pooling window, the reward is (1) 1.0 for the *top-left* feature; (2) 0.6 for the *top-right* one; (3) 0.3 for the *bottom-left*; (4) and 0 for the *bottom-right*. That is, the reward is 1.0 if the *top-left* entry is 1, regardless the values of other entries; the reward is 0.6 if *top-left* is 0 and *top-right* is 1, regardless the values of the remaining two features; the reward is 0.3 when the first 2 features are both 0 and *bottom-left* one is 1; finally, no reward if all features are 0.

The pooling regularization term is thus defined as

$$R_{\text{Pooling}} = -\beta \ln\left(\sum_{i=0}^{L\times H\times W\times C} \text{reward}(x_i)\right), \tag{5}$$

TABLE 1
Structures of the Networks Used to Evaluate O3BNN-R

Network	Network Structure	Dataset	Input Size	Categories
VGG-like	(2x128C3)-MP2-(2x256C3)-MP2-(2x512C3)-MP2-(2x1024FC)	*Cifar-10*	$32 \times 32 \times 3$	10
AlexNet	(64C11/4)-MP3-(192C5)-MP3-(384C3)-(256C3)-(256C3)-MP3-(2x4096FC)	*ImageNet*	$224 \times 224 \times 3$	1000
VGGNet	(2x64C3)-MP2-(2x128C3)-MP2-(3x256C3)-MP2-(3x512C3)-MP2-(3x512C3)-MP2-(2x4096FC)	*ImageNet*	$224 \times 224 \times 3$	1000
VGG-like-FINN [33]	(2x64C3)-MP2-(2x128C3)-MP2-(2x256C3)-MP2-(2x512FC)	*Cifar-10*	$32 \times 32 \times 3$	10

512FC refers to a fully-connected layer with 512 neurons. 2x128C3 refers to 2 convolution layer with 128 output channels and 3x3 filter. MP2 refers to a 2x2 max-pooling layer.

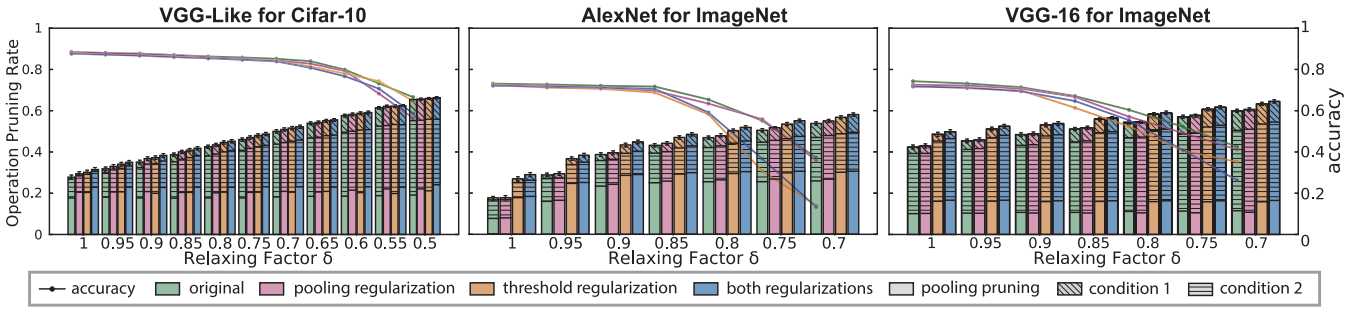


Fig. 10. Pruning rate versus Accuracy trade-off with different relaxing factors. When the relaxing factor is 1, pruning is lossless. Green lines and bars are for original models; pink, orange and blue lines and bars are for models trained with different combinations of regularization techniques.

where L is the number of pooling layers, $H \times W$ is the size of output feature maps for the pooling layer, C is the number of channels, *reward*(x_i) is the reward value reward determined during the evaluation of a pooling window, and β is a scaling factor which can be adjusted to find the best point of the trade-off between pruning benefit and accuracy loss. Since the weights of BNN models are discrete, we adopt the $\ln$ function to make the regularization function more smooth.

Overall, in the O3BNN-R training/inference co-design, we add two regularization terms to the original loss function during training to introduce more opportunities for threshold-based pruning and pooling-based pruning

$$\text{loss} = \text{loss}_{\text{network}} + R_{\text{Threshold}} + R_{\text{Pooling}}.$$

6 Evaluation and Experimental Results

In this section, we evaluate the efficiency of O3BNN-R by showing two trade-offs and comparing O3BNN-R with state-of-the-art FPGA, GPU, and CPU implementations. First, we discuss the trade-off of pruning rates versus network accuracy by adjusting the relaxing factor of thresholds, δ (Section 4.4.1). Second, we give the trade-off of hardware resource demand versus performance by adjusting the OoO capability of O3BNN-R (Section 4.4.2). Finally, using the most efficient OoO and, in the case where lossy pruning is used, the optimal relaxing factors obtained from trade-off analysis, the efficiency of O3BNN-R is compared with the state-of-the-art BNN implementations of FPGAs, GPUs, and CPUs.

We use PyTorch to build and train our BNN model. Each network is trained under 4 different configurations: without any regularization, only with pooling regularization, only with threshold regularization, and with both.

The trained models are evaluated for profiling the ideal pruning rates and measuring model accuracy, i.e., the inference accuracy on the testing set. Performance, hardware demand, and energy efficiency of O3BNN-Rs are evaluated on an embedded-scale FPGA development kit, Xilinx ZC706, which is one of the most widely used platforms in embedded systems, robotic control, autonomous cars, and research prototyping [3], [33]. In order to show the scalability of the proposed design, all O3BNN-Rs used for evaluation are equipped with 512 PEs. The FPGA results are compared with two Intel CPUs (Xeon-E5 2640 [24] and Xeon-Phi 7210 accelerator [16]), two NVIDIA GPUs (Tesla-V100 and GTX-1080 [16]), and three FPGA designs: FINN [33], ReBNet [13], and FP-BNN [24]. As for network models and datasets, we use the well-known AlexNet and VGG-16 for ImageNet and a widely used VGG-like model ([3], [16], [24], [33]) for Cifar-10. The network structures are listed in Table 1. Since FINN adjusts the structure of the VGG-like network, to make a fair comparison we use the same network as FINN (i.e., VGG-Like-FINN) as listed in the last row of Table 1.

6.1 Ideal Pruning Rate Versus Network Accuracy

Recall there are three types of edge pruning in this work: Conditions 1 and 2 of threshold-based edge pruning and Pooling Pruning. Fig. 10 shows the overall pruning rates of networks with and without regularization in training with a breakdown of the three types of pruning and network accuracy using different relaxing factors. For each network 5 k pictures selected at random are used to profile the average pruning rates.

For lossless pruning with the non-regularized models (Green bars and lines), i.e., $\delta = 1$, for VGG-Like the top-1 accuracy is 88.5 percent and the pruning rate is 27 percent. For AlexNet and VGG-16 the top-5 accuracies are 72.7 and 75.5 percent, respectively, while the pruning rates are 19 and 42 percent. When the relaxing factor is decreased, (1) the pruning rates increase almost linearly, especially for VGG-16 and VGG-Like; (2) For all networks observed so far

the accuracy remains nearly unchanged as the relaxing factor is decreased up until a threshold, which differs by network, where it suddenly decreases substantially (by many times the change before the threshold). We refer to this threshold as the *inflection point*.

The reasons of this observation are as follows. (1) When the relaxing factor is larger than the inflection point, then the relaxed threshold causes more curves to get pruned (for each neuron - compared with lossless pruning), but the threshold is not relaxed enough to change the value of neurons. Hence the network accuracy is not affected significantly. The outlier is AlexNet with large relaxing factors ($\delta > 0.9$). When the relaxing factor used in AlexNet decreases from 1 to 0.9, neurons start to flip from 0 to 1, leading to increased occurrence of Condition 1 and pooling pruning, but without hurting the accuracy; this does not happen in the other two networks. This difference comes from the old-fashioned 3×3 max-pooling filter and 11×11 CONV filter used in AlexNet. (2) When the relaxing factor is smaller than the inflection point, then the neurons' values start to flip and so incurring errors.

The relaxing factors at the inflection points of VGG-Like, AlexNet, and VGG-16 are 0.7, 0.85, and 0.9, respectively. The pruning rates of these three networks at their corresponding inflection points increase to 49, 46, and 48 percent, with only 3.3, 0.9, and 2.9 percent loss of accuracy, respectively. The networks of ImageNet are more sensitive to lowering the relaxing factors. The reason is that ImageNet has 1,000 classification categories, while Cifar-10 only has 10. The complexity of the classification task affects the vulnerability of networks, i.e., the networks' sensitivity to threshold relaxing. VGG-16 is more sensitive than AlexNet. A possible reason is that the pruning rate of VGG-16 without threshold relaxing is already close to that at the inflection point of AlexNet. We also measure the variance in pruning rates among all test images. Error bars are shown at the tops of the pruning rate bars. It is observed that for different images the pruning rates are stable.

The red, yellow, and blue bars and lines in the figure show the pruning rates and accuracy for the BNN models trained with only the threshold regularization term, only the pooling regularization term, and both. For lossless pruning, the three regularization approaches improve the pruning rates to 32, 30, and 50 percent, respectively, with accuracy loss of only 0.3, 0.6, and 0.6 percent, respectively. The two regularization terms are mostly independent; this is because they operate on different components of the BNN network.

With the relaxing factor further increased, the pruning rate increases linearly, but more slowly than the original model without regularization. We therefore observe that better pruning rates can be achieved by regularization for a relatively larger relaxing factor than a smaller one. By combining regularization and the relaxing methods, we observe an inflection point. When the relaxing factor is larger (i.e., less relaxed) than the inflection point, regularization only incurs a very small accuracy loss. However, when the relaxing factor is smaller than the inflection point (more relaxed), regularization leads to considerable accuracy loss. The inflection points of VGG-like, AlexNet and VGG-16 are 0.7, 0.85, and 0.9, respectively, the corresponding pruning rates at these inflection points are 52, 49, and 53 percent.

To show the effect of different relaxing factors, all layers share the same relaxing factor. In practice, each layer can use a different relaxing factor. For the first two CONV layers and the last three FC layers, the relaxing factor can be relatively large because errors in these layers affect the final classification result more seriously; the CONV layers in the middle can use small relaxing factors. By doing so, O3BNN-R can obtain higher pruning rates with less accuracy loss.

Fig. 11 is similar to Fig. 10 but shows the pruning rates of the three types of edge pruning at each layer with the models trained without regularizations. It is observed that, for all networks, pooling pruning is the most significant pruning type. Condition 2 is triggered much more frequently than Condition 1 at most of the layers, especially when the relaxing factor is close to 1. It is also observed that the pruning rates of the FC layers are very low when the relaxing factor is close to 1; however, those rates increase much more rapidly than the ones at the CONV layers when the relaxing factor decreases.

6.2 Hardware Demand Versus Performance

By pruning the BNN network dynamically, O3BNN-R is expected to provide better performance than a traditional accelerator with no pruning. To evaluate the efficiency of O3BNN-R, we take the classic BNN inference implementation (described in Section 3.1) as the baseline and compare it with three O3BNN-R designs with different OoO capabilities. In the baseline design, Loops 1, 2, and 3 (in Fig. 3) are processed sequentially, while Loops 4, 5, 6, and 7 are processed in parallel. The architecture of our baseline design is traditional and similar to the ones used in [24], [27]. At each clock cycle, each PE calculates the value of one curve. Assuming there are $N_{OC} \times N_{IC}$ PEs, at each cycle N_{OC} neurons with the same coordinate are completely evaluated. After $Width \times Height$ cycles, a layer is processed completely and processing begins on the next layer.

This baseline design is standard and widely used in the DNN literature. For a fair comparison of hardware consumption and performance, the baseline design is equipped with the same number of PEs as O3BNN-R. Compared with the O3BNN-R architecture, the baseline design has similar DSF and simpler PEs: they do not have logic to support pruning and OoO processing, e.g., the circular communication network and pending buffer for horizontal rotation, comparators for redundancy check, and control logic for OoO scheduling and edge pruning. Also, there is no Scoreboard in the baseline design, which uses static in-order scheduling.

The O3BNN-Rs are also compared with respect to ideal performance, i.e., the performance of an ideal system that is able to exploit all pruning opportunities profiled in Section 6.1, and without any bubbles in the pipeline incurred by dynamic scheduling. The performance differences between the ideal performance and the O3BNN-Rs indicate the OoO processing efficiency of the proposed architecture. For each O3BNN-R design, two performance values are given: one for lossless pruning and the other one for lossy pruning using the relaxing factor at the inflection point in Fig. 12.

In Fig. 12, the blue and orange lines indicate the latencies using lossless and lossy pruning, respectively. We use non-regularized models as examples to evaluate the pruning efficiency of the O3BNN-R architectures. As for the regularized models, their hardware resource demands are the same as the ones for original models and their performances

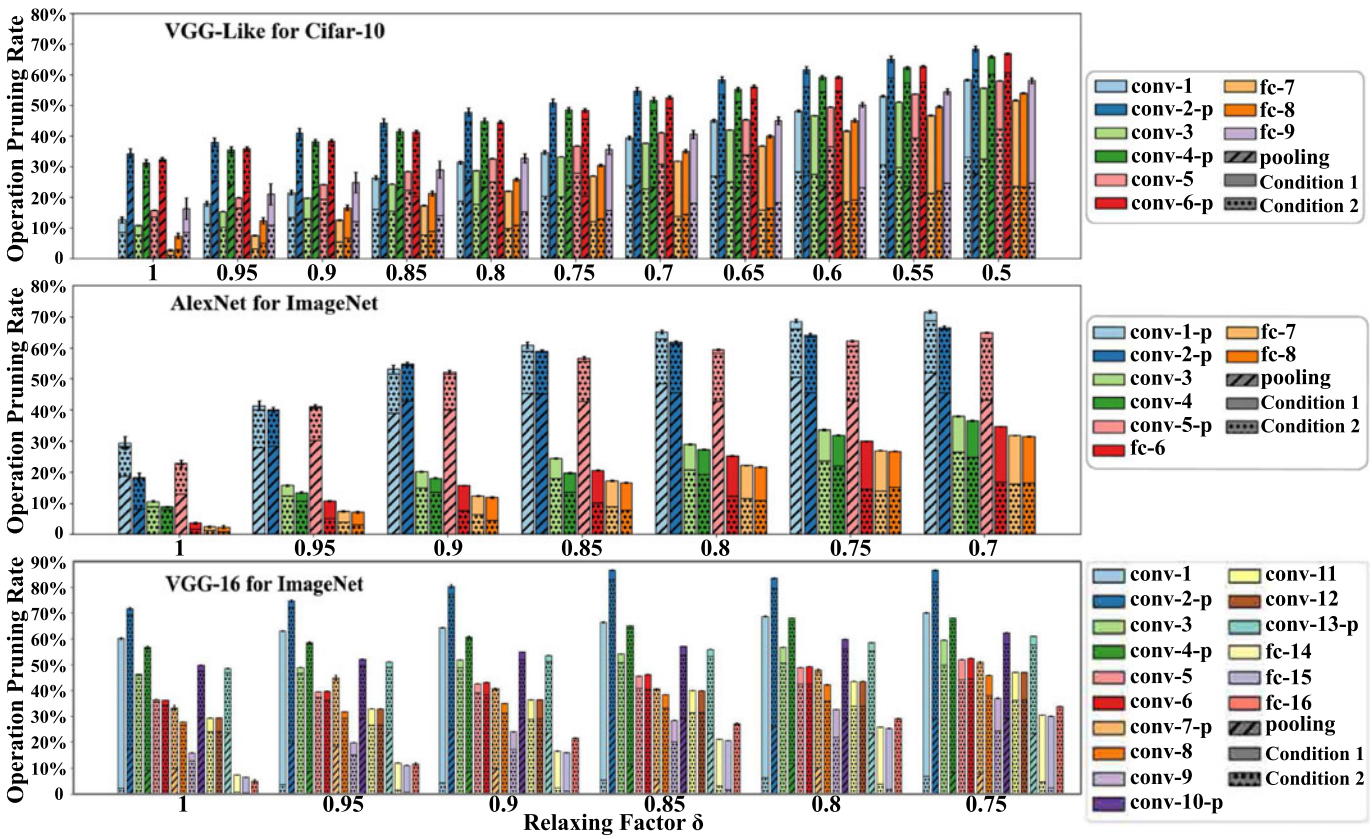


Fig. 11. Pruning rates at different layers of the non-regularized models with different relaxing factors. When relaxing factor is 1, threshold relaxing is disabled and pruning is lossless. $conv\text{-}l\text{-}p$ refers to the lth CONV layer followed by max-pooling. $fc\text{-}l$ refers to the lth fully connection layer.

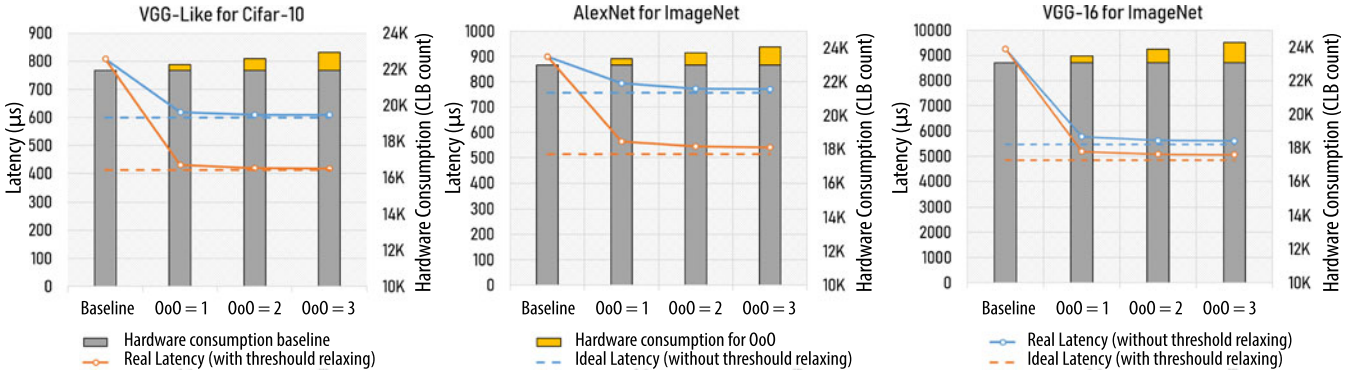


Fig. 12. Performance and hardware consumption of O3BNN-Rs with different OoO capabilities and with lossless (without threshold relaxing) or lossy (with threshold relaxing) pruning for the non-regularized models. 512-PE O3BNN-Rs are compared to 512-PE baseline without pruning and ideal design with theoretically perfect pruning. The relaxing factors for lossy pruning used in VGG-Like, AlexNet and VGG-16 are 0.7, 0.85, and 0.9.

are listed in Tables 3, 4, 5, and 6. Without any pruning, the inference latencies of VGG-Like, AlexNet, and VGG-16 are 809, 899, and 9,251 μs, respectively. The hardware consumption is 21930, 23005, and 23056 configurable logic block (CLBs). Using a O3BNN-R design whose OoO capability is 1, i.e., the Scoreboard can track the status of $1 \times N_{OC}$ curve-group at a time, the inference latencies of these three networks are decreased to 619, 793, and 5779 μs when using lossless pruning ($\delta = 1$), and 430, 565, and 5186 μs when relaxing factors at the inflection points are used. The hardware overheads are only 1.5, 1.5, and 1.8 percent compared with the baseline design. The performance of lossless and lossy O3BNN-Rs whose OoO capability is 1 are, on average, only 4.4 percent, and 6.5 percent lower than the ideal ones. The difference between ideal performance and the performance of lossy O3BNN-R is larger than the one between ideal and lossless O3BNN-R. The reason is that the pruning rates using lossy pruning are much larger than the ones using lossless pruning, requiring more OoO capability.

When the OoO capability of O3BNN-R increases from 1 to 2, for lossless pruning, the latencies are reduced by 1.5, 2.5, and 2.6 percent, respectively for the three networks, and reach 609, 774, and 5626 μs. The hardware demand is slightly increased, i.e., by 1.5, 1.5, and 1.7 percent. For lossy

TABLE 2
Latency, Hardware Demand, and Accuracy of Baseline and O3BNN-Rs With Different OoO Capabilities (1, 2, 3) and With Lossless or Lossy Pruning for Non-Regularized Models

		VGG-Like			AlexNet			VGG-16		
	OoO capability	1	2	3	1	2	3	1	2	3
Without Threshold Relaxing	Latency(μs)	619	609	608	793	774	772	5779	5626	5603
	Hardware Demand (CLB)	22264	22607	22954	23357	23736	24102	23466	23876	24285
	Accuracy	88.5%			72.7%			74.3%		
With Relaxing Factor at inflection points	Latency(μs)	430	419	418	565	545	541	5186	5080	5059
	Hardware Demand (CLB)	22264	22607	22954	23357	23736	24102	23466	23876	24285
	Accuracy	85.2%			71.8%			71.4%		
Baseline	Latency(μs)	809			899			9251		
	Hardware Demand (CLB)	21930			23005			23056		
	Accuracy	88.5%			72.7%			74.3%		

Both the baseline design and O3BNN-R implementations are equipped with 512 PEs. For lossy pruning the relaxing factors used in VGG-Like, AlexNet, and VGG-16 are 0.7, 0.85, and 0.9, respectively.

TABLE 3
Cross-Platform Evaluation of Latency, Energy Efficiency, and Accuracy: VGG-Like-FINN [3] for Cifar-10

	Existing work	O3BNN-R (without Regularization)		O3BNN-R(with both Regularizations)	
Network	VGG-Like-FINN				
Platform	FPGA ZC706 [3]	FPGA ZC706			
Frequency (MHz)	200				
Latency (μs)	283	167 (lossless)	116 (lossy)	153 (lossless)	107 (lossy)
Energy (Img/kJ)	3.9E5	6.65E5	9.58E5	7.16E5	10.38E5
Accuracy	80.1%	82.6%	79.3%	82.1%	78.5%

TABLE 4
Cross-Platform Evaluation of Latency, Energy Efficiency, and Accuracy: VGG-Like [4] for Cifar-10

	Existing works [24] or self-implemented reference [23]		O3BNN-R (without Regularization)		O3BNN-R (with both Regularizations)	
Network	VGG-Like					
Platform	Xeon E5-2640 [24]	GPU V100 [23]	FPGA ZC706			
Frequency (MHz)	2.5K	1.37K	200			
Latency (μs)	1.36E6	994	609 (lossless)	419 (lossy)	563 (lossless)	388 (lossy)
Energy (Img/kJ)	7.79	5543	1.82E5	2.65E5	1.99E5	2.83E5
Accuracy	86.31%	89.9%	88.5%	85.2%	88.2%	83.9%

pruning with the relaxing factors at the inflection points, the latencies are reduced by 2.6, 3.7, and 2.1 percent and reach 419, 545, and 5080 μs. The performance of O3BNN-R with OoO capability of 2 is, on average, only 5 percent lower than the ideal. When the OoO capability is increased from 2 to 3, there is almost no further performance improvement, but the hardware demand increases on average by 1.6 percent.

The latency, hardware demand, and accuracy of baseline and O3BNN-R-based BNN implementations are summarized in Table 2. As stated in Section 4.4, a larger Scoreboard can track the processing status of more curve-groups. The more unbalanced the pruning timing of different curve-groups, the larger the Scoreboard that is needed to avoid pipeline bubbles caused by a fully occupied Scoreboard. According to the experimental results, the support of OoO processing of $2 \times N_{OC}$ curve groups is already sufficient to unbalance the edge pruning timing.

In addition to the pipeline bubbles incurred by dynamic scheduling, another potential reason for the performance gap between the theoretical shortest latency (i.e., ideal latency) and the actual measured latency of O3BNN-R is the difference of the pruning granularity. The pruning granularity in the profiling of Section 6.1 is per-edge, which is finer than the pruning granularity of the O3BNN-R implementation, which is by curve or $K \times K$ edges. Thus, for each neuron, the real implementation may compute at most $K \times K - 1$ extra edges than the profile, leading to at most $1/N_{IC}$ extra overhead. Considering that N_{IC}s are usually larger than 100, this overhead is very small. We take this granularity overhead into consideration when evaluating O3BNN-Rs, but because of their small size they are not specifically marked in Fig. 12.

6.3 Cross-Platform Evaluation

In Tables 3, 4, 5, and 6, O3BNN-R's performance, energy efficiency, and accuracy (with lossless and lossy pruning) are compared with existing and self-implemented systems using various CPUs, GPUs, and FPGAs to accelerate BNN inference of VGG-like, VGG-like-FINN, AlexNet, and VGG-16. The performance is evaluated by using the latency of single-image inference. The energy efficiency is evaluated with respect to image inferences per Kilo-J (Image/kJ). Based on the trade-off analysis of Sections 6.1 and 6.2, the OoO capability of O3BNN-Rs is set to 2. For lossy pruning where threshold relaxing is enabled, the relaxing factors, δ, applied in the inference of VGG-Like, VGG-Like-FINN, AlexNet and VGG-16 are 0.7, 0.7, 0.85, 0.9, respectively; i.e., the δ at inflection points of accuracy lines in Fig. 10. For lossless pruning, threshold relaxing is not enabled ($\delta = 1$).

We compare O3BNN-Rs with some recently published well-known FPGA-based BNNs. Compared with FINN [3], [33], using the same network model (i.e., VGG-Like-FINN as shown in the last row of Table 1), training strategy (SGD without the proposed regularizations), and FPGA board

TABLE 5
Cross-Platform Evaluation of Latency, Energy Efficiency, and Accuracy: AlexNet [22] for ImageNet

	Existing works [13], [24] or self-implemented reference [23]			O3BNN-R (without Regularization)		O3BNN-R(with both Regularizations)	
Network	AlexNet						
Platform	GPU V100 [23]	FPGA VCU108 [13]	FPGA Stratix-V [24]	ZC706 (Stratix-V and VCU108 have 4x and 2.5x more hardware resources)			
Freq (MHz)	1.37K	200	150	200			
Latency (μs)	2226	1920	1160	774 (lossless)	545 (lossy)	661 (lossless)	510 (lossy)
Energy (Img/kJ)	2475	2.7E4	3.3E4	1.44E5	2.04E5	1.67E5	2.13E5
Accuracy	71.2%	N/A	66.8%	72.7%	71.8%	72.1%	70.6%

TABLE 6
Cross-Platform Evaluation of Latency, Energy Efficiency, and Accuracy: VGGNet-16 [30] for ImageNet

	Existing works		O3BNN-R (without Regularization)		O3BNN-R(with both Regularizations)	
Network	VGG-16					
Platform	CPU Xeon Phi 7210 [16]	GPU GeForce GTX1080 [16]	FPGA ZC706			
Frequency (MHz)	1.3K	1.61K	200			
Latency (μs)	1.18E4	1.29E4	5626 (lossless)	5080 (lossy)	4877 (lossless)	4603 (lossy)
Energy (Img/kJ)	395	433	1.97E4	2.19E4	2.23E4	2.37E4
Accuracy	76.8%	76.8%	74.3%	71.4%	73.7%	70.1%

(i.e., ZC706), lossless and lossy O3BNN demonstrate 167 μs and 116 μs single-image inference latency, corresponding to speedups of 1.69× and 2.44×. For accuracy, the lossless approach is 2.5 percent better than FINN, while the lossy approach is only 0.8 percent lower than FINN.

AlexNet results of O3BNN-R are compared to FP-BNN [24] and ReBNet [13]. The latencies of AlexNet inference accelerated by O3BNN-R with lossless and lossy pruning are 774 and 545μs. Compared to FP-BNN, lossless and lossy O3BNN-Rs are 1.50× and 2.13× faster. Compared to ReBNet, lossless and lossy O3BNN-Rs are 2.48× and 3.52× faster. Note that the FPGA boards used in FP-BNN (Stratix-V) and ReBNet (VCU108) are high-performance FPGA and have hardware resources around 4× and 2.5× as much as the one (ZC706, an embedded-FPGA) used in our evaluation. With regard to energy efficiency, lossless and lossy O3BNN-Rs are 4.4× and 6.2× better than FP-BNN and 5.3× and 7.6× better than ReBNet. The accuracy of our lossless and lossy AlexNet implementations are both higher than the one reported in FP-BNN. The accuracy of AlexNet is not reported in ReBNet.

We also measure the inference latency of VGG-16 for ImageNet, which is 5.6ms and 5.1ms for a single image using lossless and lossy O3BNN-Rs, respectively. The accuracy is 74.3 and 71.4 percent. Compared with CPUs and GPUs, the latencies for lossless and lossy O3BNN-R are 47.7 and 43.1 percent of the latency for Xeon Phi 7210 [16], and 43.6 and 39.4 percent of the latency for GTX1080 [16], with similar accuracy. As the FPGA board in this work is for embedded applications, the energy advantage is even more prominent. The energy efficiency of O3BNN-Rs is 50× and 55× better than that of the Xeon-Phi 7210 and 45× and 51× better than the GTX-1080 for lossless and lossy pruning designs. The comparisons of other networks and with other CPUs and GPUs are listed in Tables 4 and 5.

By applying the proposed regularization techniques, the pruning efficiency of O3BNN-R is further improved with almost negligible loss in accuracy. Compared with existing work, regularized lossless and lossy O3BNN-R achieve 1.85× and 2.64× speedups over FINN, 1.75× and 2.27× over FP-BNN [24], and 2.90× and 3.76× over ReBNet [13]. Compared with the models without regularization, the improvements are, on average, 15 percent for lossless pruning and 8 percent for lossy pruning.

7 Related Work

BNNs have been implemented variously [10], [11], [13], [16], [23], [24], [27], [33], [37]. Because of the flexibility and direct bit-manipulation capability of FPGAs [9], [12], [29], most BNN implementations are FPGA-based [13], [24], [27], [33], [37]. We have already discussed FINN [33] in Section 2. In [37], Zhao, *et al.*, proposed the first high-level-synthesis-based BNN implementation on FPGAs. In [24], Liang, *et al.*, proposed an FPGA-based BNN accelerator that drastically cuts down the hardware consumption by using resource-aware model analysis. Recently a CPU-based BNN design was proposed [16] that relies on bit-packing and AVX/SSE vector instructions to achieve good bit-processing performance. All of these are static designs and none takes advantage of the pruning opportunities of BNNs.

With regard to the pruning of BNNs, multiple studies have described BNN edge and neuron pruning. We have already discussed the neuron pruning work [6] in Section 2. In [21], Li, *et al.*, proposed a new training method for BNNs in which bit-level accuracy sensitivity analysis is conducted after initial training. The channels with low accuracy sensitivity are then pruned. These pruning methods are all performed offline and before inference. During inference, the designs are entirely static. Also, the network accuracy is often compromised due to the pruning of neurons or edges. Our method–in contrast to the static and offline pruning approaches–is dynamic with on-line pruning of inference at run-time. Without a relaxing factor, this method can prune a large number of edges without affecting the accuracy of the networks.

Compared with the studies published on CNN pruning [14], [26], [36], the design here has three distinguishing aspects: (1) Run-time dynamic pruning for post-training network models; (2) Without compromising accuracy, no fine-tuning at training process and no need to retrain models; (3) 2D-rotative OoO-architecture to handle irregular parallelism from run-time dynamic pruning.

8 Discussion

Generality of O3BNN-R. O3BNN-R is generally useful for any Quantized Neural Networks (QNNs) but is especially efficient when the QNN's feature data-width is $\leq$ 4-bit. For a QNN layer with q-bit features, each output channel has at

least $2^q - 1$ thresholds. To check the triggering condition of threshold-based pruning, partial accumulation results need to be compared to q thresholds. Compared with BNNs with 1-bit features, each PE of O3BNN-R needs to perform $q - 1$ extra comparisons per cycle, leading to increased usage of computation logic. Furthermore, more thresholds need to be stored, leading to higher on-chip memory demand. Based on our experiments, this hardware cost overhead is negligible for QNNs with $\leq$ 4-bit features so O3BNN-Rs can provide significant speedups. For QNNs with wider data-widths, O3BNN-R needs to be optimized to reduce the hardware resource overhead; this is future work.

Could This Approach Work for GPUs? First, most GPUs still follow the SIMT warp- or wavefront-based execution model and thus cannot dynamically switch-in/out tasks at per-lane granularity. Second, the out-of-order capability enabled in the O3BNN-R design relies on the high flexibility of the architecture, while the execution of GPU threads is in-order. It may therefore be difficult for existing GPUs to effectively leverage dynamic pruning with its randomly occurrences.

9 Conclusion

We propose O3BNN-R, an OoO high-performance BNN inference architecture with fine-grained and dynamic pruning. The contributions of O3BNN-R are two-fold. For algorithm, O3BNN-R demonstrates the highly-condensed BNN model can be further shrunk significantly without loss on accuracy by dynamically pruning irregular redundant edges at all CONV, FC, and POOLING layers. For architecture, O3BNN-R is an out-of-order architecture which (1) checks the redundancy of operations at run-time and in a fine-grained-manner; (2) avoids these redundant operations and ceases the evaluation of a neuron in case its binary output can be determined early; and (3) schedules the evaluation workload of neurons to hardware in a 2D-rotative OoO scheduling methodology with almost perfect utilization. Furthermore, to further enhance the pruning rate, we proposed 2 regularization techniques to direct the models training towards the direction leading to more pruning opportunities in our O3BNN-R architecture. We have evaluated our design on an FPGA platform using VGG-16, AlexNet for ImageNet, and a VGG-Like network for Cifar-10. Results show that our out-of-order approach can prune 27, 19, and 42 percent of the operations for the three networks respectively, without any accuracy loss, leading to, at least, 1.7×, 1.5×, 2.1× inference-speedup over state-of-the-art FPGA/GPU/CPU BNN implementations. With only 3.3, 0.9 and 2.9 percent accuracy loss, the pruning rate increases to 49, 43, 48 percent, respectively, with, at least, 2.4×, 2.1×, and 2.3× speedup. The proposed regularization techniques can further improve the performance of O3BNN-R, on average, by 15 percent with only 0.5 percent accuracy loss. Our approach is inference runtime pruning, so no retrain or fine-tuning in training is needed. Although FPGA is used as a showcase in this paper, the proposed architecture can be adopted on any smart devices as well.

Acknowledgments

This work was supported, in part, by the US National Science Foundation through Awards CNS-1405695 and CCF-1618303/7960; by the NIH through Award 1R41GM128533; by grants from Microsoft and Red Hat; and by Xilinx and by Intel through donated FPGAs, tools, and IP. This research was also partially funded by Pacific Northwest National Laboratory's DMC-CFA and DeepScience-HPC LDRD projects. The evaluation platforms were supported by the U.S. DOE Office of Science, Office of Advanced Scientific Computing Research, under award 66150: "CENATE - Center for Advanced Architecture Evaluation." The Pacific Northwest National Laboratory is operated by Battelle for the U.S. Department of Energy under contract DE-AC05-76RL01830.

References

[1] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer CNN accelerators," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2016, pp. 1–12.

[2] J. Bethge, C. Bartz, H. Yang, Y. Chen, and C. Meinel, "MeliusNet: Can binary neural networks achieve mobilenet-level accuracy?" 2020, *arXiv: 2001.05936*.

[3] M. Blott *et al.*, "FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 3, 2018, Art. no. 16.

[4] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 3123–3131.

[5] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," 2016, *arXiv:1602.02830*.

[6] T. Fujii, S. Sato, and H. Nakahara, "A threshold neuron pruning for a binarized deep neural network on an FPGA," *IEICE Trans. Inf. Syst.*, vol. 101, no. 2, pp. 376–386, 2018.

[7] T. Geng *et al.*, "AWB-GCN: A graph convolutional network accelerator with runtime workload rebalancing," *53rd IEEE/ACM Int. Symp. Microarchit. (MICRO)*, pp. 1–15, 2020.

[8] T. Geng, T. Wang, A. Li, X. Jin, and M. Herbordt, "FPDeep: Scalable acceleration of CNN training on deeply-pipelined FPGA clusters," *TC*, vol. C-69, no. 8, pp. 1143–1158, 2020, doi: 10.1109/TC.2020.3000118.

[9] T. Geng *et al.*, "FPDeep: Acceleration and load balancing of CNN training on FPGA clusters," in *Proc. IEEE 26th Annu. Int. Symp. Field-Programmable Custom Comput. Mach.*, 2018, pp. 81–84.

[10] T. Geng *et al.*, "LP-BNN: Ultra-low-latency BNN inference with layer parallelism," in *Proc. IEEE 30th Int. Conf. Appl.-Specific Syst. Archit. Processors*, 2019, pp. 9–16.

[11] T. Geng *et al.*, "O3BNN: An out-of-order architecture for high-performance binarized neural network inference with fine-grained pruning," in *Proc. 33rd ACM Int. Conf. Supercomput.*, 2019, pp. 461–472.

[12] A. D. George, M. C. Herbordt, H. Lam, A. G. Lawande, J. Sheng, and C. Yang, "Novo-G#: Large-scale reconfigurable computing with direct and programmable interconnects," in *Proc. IEEE High Perform. Extreme Comput. Conf.*, 2016, pp. 1–7.

[13] M. Ghasemzadeh, M. Samragh, and F. Koushanfar, "ReBNet: Residual binarized neural network," in *Proc. IEEE 26th Annu. Int. Symp. Field-Programmable Custom Comput. Mach.*, 2018, pp. 57–64.

[14] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 1389–1397.

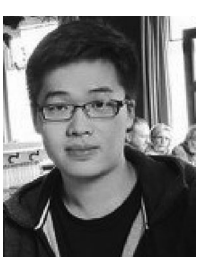
[15] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. Amsterdam, The Netherlands: Elsevier, 2011.

[16] Y. Hu *et al.*, "BitFlow: Exploiting vector parallelism for binary neural networks on CPU," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2018, pp. 244–253.

[17] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 4107–4115.

[18] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6869–6898, 2017.

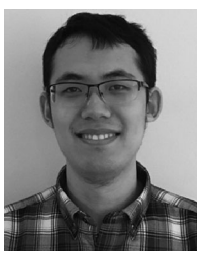
[19] S. Ji, W. Xu, M. Yang, and K. Yu, "3D convolutional neural networks for human action recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 1, pp. 221–231, Jan. 2013.
[20] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2014, pp. 1725–1732.
[21] S. Khoram and J. Li, "Adaptive quantization of neural networks," in *Proc. Int. Conf. Learn. Representations*, 2018, pp. 1–13.
[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[23] A. Li, T. Geng, T. Wang, M. Herbordt, S. L. Song, and K. Barker, "BSTC: A novel binarized-soft-tensor-core design for accelerating bit-based approximated neural nets," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2019, pp. 1–30.
[24] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, "FP-BNN: Binarized neural network on FPGA," *Neurocomputing*, vol. 275, pp. 1072–1086, 2018.
[25] L. Lu, Y. Liang, Q. Xiao, and S. Yan, "Evaluating fast algorithms for convolutional neural networks on FPGAs," in *Proc. IEEE 25th Annu. Int. Symp. Field-Programmable Custom Comput. Mach.*, 2017, pp. 101–108.
[26] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient transfer learning," 2016, *arXiv:1611.06440*.
[27] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh, and D. Marr, "Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, and ASIC," in *Proc. Int. Conf. Field-Programmable Technol.*, 2016, pp. 77–84.
[28] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-net: ImageNet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 525–542.
[29] J. Sheng, C. Yang, and M. C. Herbordt, "High performance communication on reconfigurable clusters," in *Proc. 28th Int. Conf. Field Programmable Logic Appl.*, 2018, pp. 219–2194.
[30] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Representations*, 2015, pp. 1–14.
[31] W. Tang, G. Hua, and L. Wang, "How to train a compact binary neural network with high accuracy?" in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 2625–2631.
[32] R. M. Tomasulo, "An efficient algorithm for exploiting multiple arithmetic units," *IBM J. Res. Develop.*, vol. 11, no. 1, pp. 25–33, 1967.
[33] Y. Umuroglu *et al.*, "FINN: A framework for fast, scalable binarized neural network inference," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, 2017, pp. 65–74.
[34] T. Wang, T. Geng, A. Li, X. Jin, and M. Herbordt, "FPDeep: Scalable acceleration of CNN training on deeply-pipelined FPGA clusters," *IEEE Trans. Comput.*, vol. 69, no. 8, pp. 1143–1158, Aug. 2020.
[35] X. Wei *et al.*, "Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs," in *Proc. 54th Annu. Des. Autom. Conf.*, 2017, Art. no. 29.
[36] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 5687–5695.
[37] R. Zhao *et al.*, "Accelerating binarized convolutional neural networks with software-programmable FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, 2017, pp. 15–24.
[38] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients," 2016, *arXiv:1606.06160*.



Tong Geng received the BS degree from Zhejiang University, Hangzhou, China, in 2013, and the MS degree from the Eindhoven University of Technology, Eindhoven, The Netherlands, in 2015. He is currently working toward the PhD degree with the Electrical and Computer Engineering Department, Boston University, Boston, Massachusetts. His research interests include machine learning, computer architecture, parallel computing, and reconfigurable computing.

Ang Li received the bachelor's degree from the Computer Science Department, Zhejiang University, Hangzhou, China, in 2010, and the joint PhD degrees from the Department of Electrical and Computer Engineering, National University of Singapore, Singapore and the Department of Electrical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands, in 2016. From 2010 to 2012, he worked in industry as a software developer. He is currently a research scientist in the HPC Group, Pacific Northwest National Laboratory, Richland, Washington.

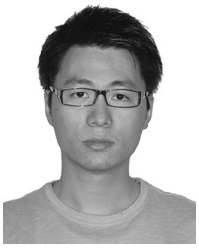
Tianqi Wang received the bachelor's degree in applied physics from the University of Science and Technology of China, Hefei, China, in 2012, where he is currently working toward the PhD degree. He was a visiting scholar with Boston University from 2017 to 2019. He is currently working on accelerating N-body problems using FPGA-centric clusters.



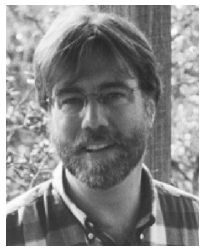
Chunshu Wu received the bachelor's degree in physics from the Dalian University of Technology, Liaoning, China, 2016, and the master's degree in physics from Brown University, Providence, Rhode Island, in 2018. He is currently working toward the PhD degree with the Electrical and Computer Engineering Department, Boston University, Boston, Massachusetts. He is currently working on FPGA-based acceleration for molecular dynamics.

Yanfei Li received the bachelor's degree from the Information Science Department, Zhejiang University, Hangzhou, China, where she is currently working toward the PhD degree. Her research interests include computer vision, machine learning, and image processing.

Runbin Shi received the BEng and MEng degrees from Soochow University, Suzhou, China, in 2013 and 2016, respectively. He has been working toward the PhD degree with the Department of Electrical and Electronic Engineering, University of Hong Kong, Hong Kong, since September 2016. His research interest is on modeling and optimization for FPGA-accelerator design.



Wei Wu received the bachelor's degree in software engineering from the Beijing Institute of Technology, Beijing, China, the master's degree in computer engineering from Purdue University, West Lafayette, Indiana, and the PhD degree in computer science from the University of Tennessee, Knoxville, Tennessee. He is currently a research scientist with Los Alamos National Laboratory. His research interests involve runtime systems and programming models for heterogeneous systems.



Martin Herbordt is a professor with the Department of Electrical and Computer Engineering, Boston University where he directs the Computer Architecture and Automated Design Lab. His group works on accelerating HPC applications, especially in molecular dynamics, bioinformatics, and machine learning, and on system aspects of FPGA clusters and clouds.