

# O3BNN: An Out-Of-Order Architecture for High-Performance Binarized Neural Network Inference with Fine-Grained Pruning

Tong Geng<sup>1,2</sup>, Tianqi Wang<sup>1</sup>, Chunshu Wu<sup>1</sup>, Chen Yang<sup>1</sup>, Wei Wu<sup>3</sup>, Ang Li<sup>2</sup>, Martin C. Herbordt<sup>1</sup>

<sup>1</sup>Boston University, Boston, MA

<sup>2</sup>Pacific Northwest National Laboratory, Richland, WA

<sup>3</sup>Los Alamos National Laboratory, Los Alamos, NM

{tgeng, tianqi, happycwu, cyang90}@bu.edu, ww@lanl.gov, ang.li@pnnl.gov, herbordt@bu.edu

## ABSTRACT

Binarized Neural Networks (BNN) have drawn tremendous attention due to significantly reduced computational complexity and memory demand. They have especially shown great potential in cost- and power-restricted domains, such as IoT and smart edge-devices, where reaching a certain accuracy bar is often sufficient, and real-time is highly desired.

In this work, we demonstrate that the highly-condensed BNN model can be shrunk significantly further by dynamically pruning irregular redundant edges. Based on two new observations on BNN-specific properties, an out-of-order (OoO) architecture – O3BNN, can curtail edge evaluation in cases where the binary output of a neuron can be determined early. Similar to Instruction-Level-Parallelism (ILP), these fine-grained, irregular, runtime pruning opportunities are traditionally presumed to be difficult to exploit. We evaluate our design on an FPGA platform using three well-known networks, including VggNet-16, AlexNet for ImageNet, and a VGG-like network for Cifar-10. Results show that the out-of-order approach can prune 27%, 16%, and 42% of the operations for the three networks respectively, without any accuracy loss, leading to at least 1.7×, 1.5×, and 2.1× speedups over state-of-the-art BNN implementations on FPGA/GPU/CPU. Since the approach is inference runtime pruning, no retraining or fine-tuning is needed. We demonstrate the design on an FPGA platform; however, this is only for showcasing the method: the approach does not rely on any FPGA-specific features and can thus be adopted by other devices as well.

## CCS CONCEPTS

• **Computing methodologies** → **Parallel algorithms**; • **Computer systems organization** → **Parallel architectures**; **Neural networks**.

## KEYWORDS

Machine Learning, BNN, High-Performance Computing, Pruning, Out-of-Order Architecture

ACM acknowledges that this contribution was authored or co-authored by an employee, contractor, or affiliate of the United States government. As such, the United States government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for government purposes only.

ICS '19, June 26–28, 2019, Phoenix, AZ, USA

© 2019 Association for Computing Machinery.

ACM ISBN 978-1-4503-6079-1/19/06...\$15.00

<https://doi.org/10.1145/3330345.3330386>

## ACM Reference Format:

Tong Geng, Tianqi Wang, Chunshu Wu, Chen Yang, Wei Wu, Ang Li, Martin Herbordt. 2019. O3BNN: An Out-Of-Order Architecture for High-Performance Binarized Neural Network Inference with Fine-Grained Pruning. In *2019 International Conference on Supercomputing (ICS '19)*, June 26–28, 2019, Phoenix, AZ, USA. 12 pages. <https://doi.org/10.1145/3330345.3330386>

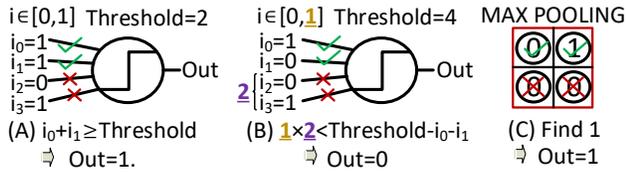
## 1 INTRODUCTION

Deep-Neural-Networks (DNNs) have been adopted widely due to their ability to learn sufficiently well to achieve high accuracy [6, 8, 16, 17]. For many high-volume but cost/power-restricted applications, however, accuracy is not an absolute requirement [15, 27]. Rather, reaching a certain well-defined level of accuracy is often sufficient, but with low-latency—or even real-time—and low-cost being highly desired. This is especially true for IoT and smart-edge devices [14, 21, 30, 33].

To satisfy these requirements, Binarized-Neural-Networks (BNNs) [24] have received tremendous attention. BNNs use a single bit to encode each neuron and parameter and thus significantly reduce computation complexity (from floating-point/integer to Boolean operations) and memory demand (from bytes to bit-per-datam for both memory storage and bandwidth). This potentially reduces delay of inference by orders-of-magnitude with acceptably small loss in accuracy.

Having only two values per neuron, a BNN's network structure is significantly different from a conventional DNN's. Such differences expose various new optimization opportunities. For example, Umuroglu, et al., [29] show that the Batch-Normalization functions (BN) in most BNNs can be simplified to a threshold-based compare operation thus avoiding the floating-point calculation. Fujii, et al., [5] 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, as well the as associated computation, is reduced. The accuracy, however, is compromised.

This work is motivated by these previous studies together with the following observation. 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 (the links between two adjacent neurons) linked to this neuron with a fixed threshold decided 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



**Figure 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).**

(Figure 1-A); or (b)  $0^1$ , as long as the current accumulation is still small enough that it has no chance of reaching the threshold in the remaining accumulation (Figure 1-B). This cessation operation is analogous to breaking out of a loop as soon as the result is determined. We call this approach **threshold-based edge pruning**.

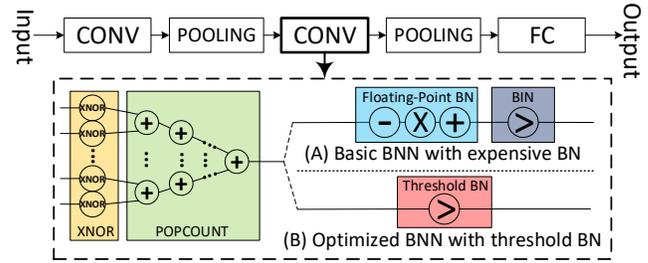
Another observation is about pooling. For max-pooling, which is the most widely used pooling method in BNNs, since the inputs are binary, in the case where one of the  $n \times n$  inputs (typically  $2 \times 2$ ) is shown to be 1, the pooling result is 1 as well; thus we can avoid the evaluation of the remaining pooling entries. For example, in Figure 1-C there is a 1 for the second entry in the  $2 \times 2$  pooling, so we can prune the convolution computation for the left two neurons. We refer to this approach as **pooling-based edge pruning**. This approach can be applied to min- and mean-pooling as well.

Although both observations are immediate, efficient harvesting of these pruning opportunities is challenging. This is because both are irregular, occasional, data-dependent, run-time, and strongly depend 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 turn out that all pooling entries are eventually 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, but we still need parallelism to guarantee high performance. On the other hand, in case the evaluation of a neuron is early-terminated, the execution gap needs to be filled instantly to avoid losing performance through pipeline bubbles. This combined challenge has (elsewhere) been considered to be very difficult [5]. In this paper, we overcome these difficulties by proposing an out-of-order edge-pruning architecture that can be used to effectively accelerate BNN inference. The main contributions 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; and

<sup>1</sup>-1 is encoded as 0 in XNOR Net[24].



**Figure 2: A typical 2-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.**

- An architecture called **O3BNN** to realize efficient run-time BNN inference pruning.

We evaluate our design on an FPGA platform using VGG-16 [26], AlexNet [19] for ImageNet and a VGG-like network [3] for Cifar-10. Evaluation results demonstrate that our out-of-order approach can prune 27%, 16%, and 42% of the operations for the three networks respectively, without any accuracy loss, bringing 1.7 $\times$ , 1.5 $\times$ , and 2.1 $\times$  inference-speedup over state-of-the-art FPGA/GPU/CPU BNN implementations.

The organization of this paper is as follows. In Section 2, BNN backgrounds and the motivations of using edge-pruning to optimize networks are discussed. In Section 3, the edge-pruning opportunities are introduced. In Section 4, an Out-of-Order BNN pruning design is proposed in detail. In Section 5, experimental results are given and discussed. In Section 6, related work is discussed. Finally, we conclude and suggest further work in Section 7.

## 2 BNNs AND MOTIVATION FOR PRUNING

### 2.1 Basic BNN Structure

BNNs evolved from conventional CNNs through Binarized Weight Networks (BWN) [3] with the observation that if the weights were binarized to 1 and  $-1$ , expensive floating-point 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. With this observation, 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 an efficient way of implementing convolutional operations via Exclusive-NOR (XNOR) and bit-counting operations [4, 24]. In this paper, we use the terminology from XNOR-Net [24].

The basic structure of BNNs has four essential functions in each CONV/FC layer: XNOR, Population Count (POPCOUNT), Batch Normalization (BN), and Binarization (BIN) (see Figure 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, which is compulsory for high accuracy in BNNs. Batch Normalization (BN) incorporates full-precision floating-point (FP) operations, i.e., two FP MUL/DIV

and three FP ADD/SUB:

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

The normalized outputs from BN (i.e.,  $y_{i,j}$ ), which are floating point, are binarized in BIN by comparing with 0:

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

Here, BIN acts as the non-linear activation function. Max pooling can be required. Traditionally, pooling is between 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, significantly reducing computation complexity.

## 2.2 Motivation for Pruning

As mentioned in Section 1, researchers have observed various opportunities to further optimize the basic BNN structure. FINN [2, 29] stands out by merging BN and BIN. As shown in Figure 2, the original FP-based BN function in Equation 1 and BIN function in Equation 2 are integrated as a threshold:

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

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

Another motivating study is about neuron pruning [5]. 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 due to the reduced number of neurons. The authors also mentioned edge-pruning, expecting it can be more beneficial than neuron pruning, but did not pursue it due to the irregularity and the difficulty in hardware implementation. In this work, we demonstrate that edge-pruning is feasible and propose a dynamic out-of-order architecture to realize it with little hardware overhead, and with no (or controllable) 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  $1 \times 1$  BCONVs.

### 3.1 Basic BCONV design

Figure 3 shows the pseudo code for a BNN with BCONV layers.  $K$  is the convolution filter size;  $\text{NIC}$  is the number of input channels;  $\text{NOC}$  is the number of output channels;  $\text{WIDTH}$  and  $\text{HEIGHT}$  are the width and height of the feature maps;  $\text{LAYER}$  is the number of layers.

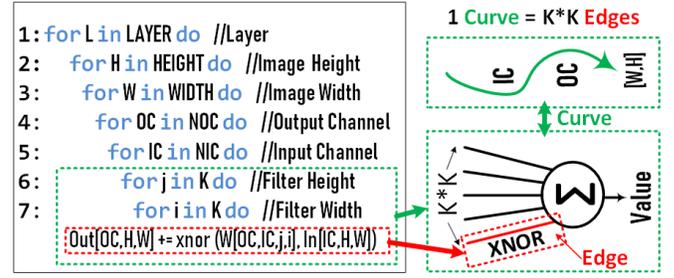


Figure 3: Pseudo code of a traditional BCONV/BFC without pruning and the symbols of edge and curve.

There are 7 loops. Each iteration of Loop 7 processes an **edge**, i.e., XNOR + POPCOUNT, in the network graph, denoted in red in Figure 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, as shown in Figure 3 and denoted in green. 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 arrow of the curve to indicate its index in Loop 2 and Loop 3. Therefore, a curve indexed by  $[\text{IC}, \text{OC}, \text{W}, \text{H}]$  represents the workload of evaluating a convolution window of  $K \times K$  neurons for the input channel IC and the output channel OC at location  $[\text{W}, \text{H}]$  of the input feature map. The complete calculation of each output channel requires the accumulation of  $\text{NOC}$  curves (Loop-4); the entire BCONV layer requires  $\text{NIC} \times \text{NOC} \times \text{HEIGHT} \times \text{WIDTH}$  curves. In our design *a curve is the basic granularity for edge-pruning*. Existing work [5, 23, 32] generally exploits parallelism in the loop-nest through:

- Loop 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.
- Loop 2-3: Generally in sequential for the sake of data reuse across neighboring  $[\text{H}, \text{W}]$  (i.e., neighboring CONV windows overlap).
- Loop 1: Mostly sequential, as for data parallelism. In general, 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

Recall the threshold-based BN for each output channel (OC in Loop 4). Figure 4-A illustrates the process: (1) calculate and accumulate  $\text{NIC}$  curves for this output channel, i.e., Loop 5; (2) binarize via threshold comparison. Since the threshold is fixed in inference and the output is a binary value, we may 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** to denote the accumulation results for  $\text{ACC\_Cur}$  curves; and **T** to denote threshold.

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

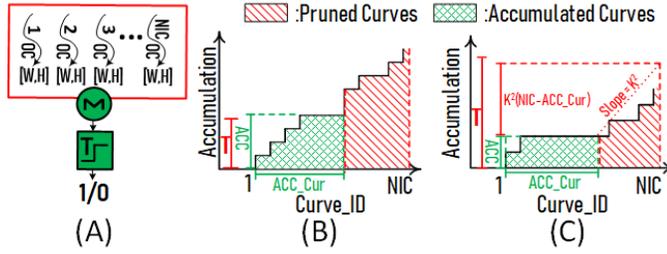


Figure 4: (A): illustration of the evaluation process of an output neuron using threshold-based BN function; (B)&(C): Conditions of threshold-based edge pruning

weights in BNNs are binary (1/0), the curve’s value is always non-negative. Therefore accumulation will never decrease  $ACC$ . Consequently, whenever  $ACC$  exceeds  $T$ , the binarization result is 1 and will never flip to 0 during the remaining accumulation. Therefore  $(NIC - ACC\_Cur)$  curves are pruned, as shown in Figure 4-B.

**Condition 2:**  $ACC < T - K^2 \times (NIC - ACC\_Cur)$  implies that  $(NIC - ACC\_Cur)$  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  $(NIC - ACC\_Cur)$  curves the maximum possible contribution is  $K^2 \times (NIC - ACC\_Cur)$ . Therefore if  $ACC + K \times K \times (NIC - ACC\_Cur)$  is still less than  $T$ , then  $ACC$  will never reach  $T$  and we can safely prune the remaining  $(NIC - ACC\_Cur)$  curves and output 0 (as shown in Figure 4-C).

**Implementation Difficulty:** (1) To prune in Loop 5, Loop 5 must execute sequentially; this may harm parallelism that can be otherwise be leveraged in a traditional design; (2) Due to pruning, the latency for each iteration in Loop 4 can differ dramatically, which may lead to a workload imbalance; (3) such dynamic, asynchronous, and data-dependent slacks must be immediately filled and sufficiently leveraged, 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 constrained.

### 3.3 Pooling-based-Edge-Pruning

Given a threshold-based BN design, we now consider the pooling function (see Figure 5-A). Figure 5-B shows how the 4 entries of a  $2 \times 2$  pooling window are sub-sampled after a convolution. As the entries are binary, the “max” operation is equivalent to a bitwise-OR among the 4 entries. Therefore, once an entry is identified as 1 (e.g., the first entry in Figure 5-B), the pooling result is 1; we can safely prune the evaluation of the remaining entries. For example, the convolution of 3 entries in Figure 5-B) is pruned.

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

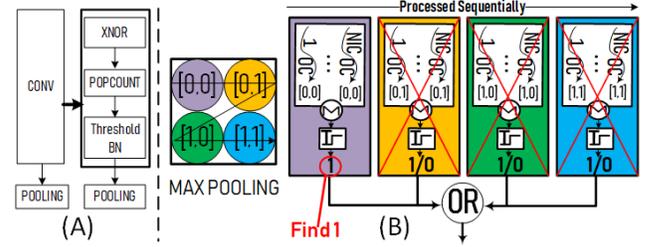


Figure 5: (A): BNN structure used in this work: POOLING follows threshold-based BN; (B): Illustration of the condition of Pooling-based Edge Pruning.

## 4 OUT-OF-ORDER BNN PRUNING DESIGN

Faced with the conflict between sequential (for pruning) and parallel (for performance) execution, 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 hardware implementation.

### 4.1 Parallelization Strategy

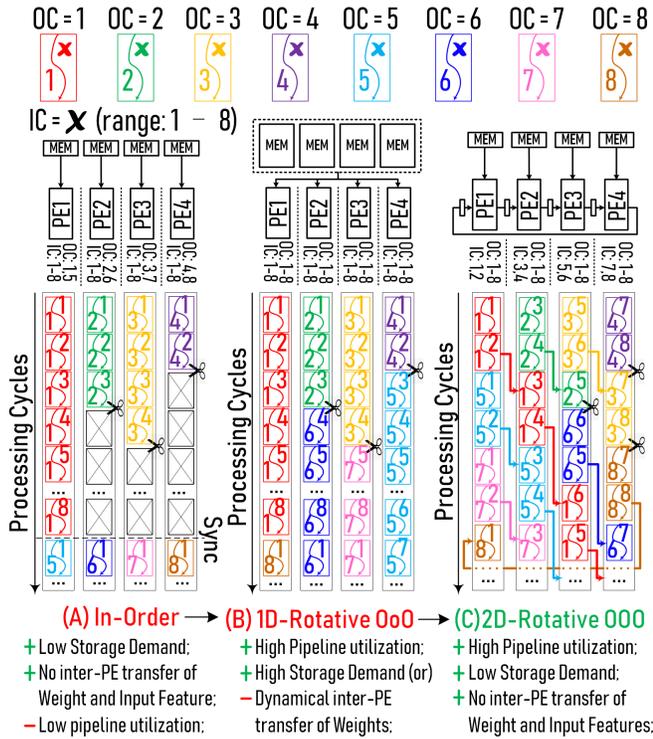
To achieve threshold and pooling edge-pruning, Loop 5 must be executed sequentially and Loop 4 partially sequentially. To compensate for this reduced parallelism, we exploit the inter-layer parallelism (i.e., model parallelism) from Loop 1. Note, the data reuse in Loops 2 and 3 is still critical for performance. Here, we resolve layer-wise workload imbalance by allocating computation resources proportionally with per-layer workload. For large storage demand, we adopt a *layer-fusion* technique, as referred to in [1]. Overall, parallelism from  $K$  (Loop 6-7),  $OC$  (Loop 4), and  $L$  (Loop 1) are exploited for parallel execution.

### 4.2 Rotative Workload Scheduling

For the clarity and without loss of generality, let us first assume that 4 Processing Elements (PEs) process a BNN layer with 8 input channels ( $IC=8$ ) and 8 output channels ( $OC=8$ ). We present three approaches to show the evolution of our 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 (see Figure 6-A). The simple in-order design has two advantages. The first is low storage demand. Since the  $NOC$  output channels can be statically partitioned among PEs (e.g., PE2 in Figure 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 a fixed curve mapping plan. The drawback of in-order scheduling, however, is that it hardly benefits from pruning (except when all PEs conduct pruning, which is rare), wasting computation resources and leading to 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  $OC=2$



**Figure 6: 3 methods of workload scheduling. (A) is a 2D-Rotative OoO scheduling method and is adopted in O3BNN.**

(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 design: we use a counter to rotatively count from 1 to 8 for IC, while dynamically controlling the value of OC for OoO. As shown in Figure 6-B, PEs periodically execute curves with IC rotating from 1 to 8, while OC is updated in the arbitrary cycle in case pruning occurs. We refer to the group of curves with the same OC (but different IC) as a **curve-group**.

This design achieves OoO for pruning 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 has to retain a local copy of the entire set of weights. If the weights were globally shared, a very clever data-feeding circuit would have to be designed for issuing the required weight portion on-time. This would introduce delay as well as area and power overhead. In addition, 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 in a time-sharing approach. Specifically, rather than partitioning curves along OC among PEs (as for the in-order design), we partition along ICs. In other words, each PE statically handles a portion of ICs. For example, PE3 in Figure 6-C only processes IC-5 and IC-6. Consequently weights can also be statically partitioned along IC and distributively reside in PE’s local memory. For the horizontal

rotation, PEs are connected to their right neighbors forming a unidirectional circle among PEs (horizontally in Figure 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 for execution each cycle, a PE first checks its left side buffer and continues the unfinished curve-group; otherwise, it fetches a new curve-group (the current curve-group is either pruned or completed) and starts execution.

To summarize: by simultaneously rotating them along the vertical dimension, we dynamically dispatch curve-groups with desired pruning capability and with low memory addressing cost; and 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 adopted for the O3BNN hardware implementation.

### 4.3 O3BNN Architecture

We introduce O3BNN architecture as shown in Figure 7. To achieve workload balancing, the PEs and other hardware resources are allocated roughly proportionally to workload per-layer. 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 3 major modules: *PE array* for workload execution, *Score-Board* for tracking curve execution status and ensuring in-order commitment, and *Data Feeding System*, or DFS, for buffering and feeding correct input data.

**4.3.1 Processing Element Array.** Figure 8 shows the detailed architecture of the PE array. To realize horizontal rotation, PEs are linked via a unidirectional circular communication network with two channels: one for forwarding the unfinished curve-group (red-line) and one for conveying the present ACC value. Inside a PE, there are three buffers. (1) The buffer at the bottom-left is used to buffer 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 among reused input features for the next OC (i.e., curve-group), or buffering a new input feature from the next image pixel  $[H, W]$ . The FIFO loop-back implements vertical rotation by repeatedly reusing the buffered data for different OCs. (2) The middle buffer is for distributively storing weights with each PE holding  $NIC/PEs * NOC$  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 completely new curve-group; (b) self-input for continuous 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 from continuously processing its curve-group portion or conveying the curve-group to its right neighbor PE.

**4.3.2 Scoreboard.** Analogous to reservation stations in Tomasulo’s algorithm [12, 28] 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 right order. As shown in Figure 9, each entry tracks a curve-group and has three basic fields: OC for curve-group ID, status for control, and the 1-bit

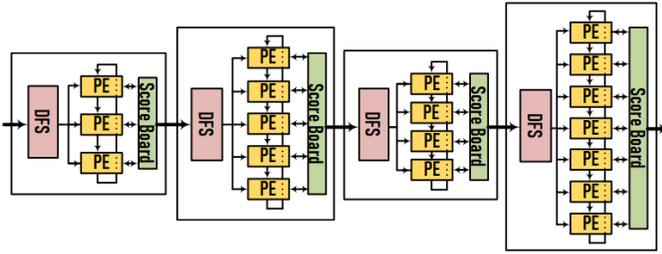


Figure 7: Overall architecture of O3BNN; architectures of PE array, Scoreboard and DFS are shown in Figure 9, 8 and 10.

output for this OC. Each column with  $NOC$  entries in the same color tracks  $NOC$  curve-groups for the pixels with the same  $[W, H]$  at all ( $NOC$ ) output 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  $NOC$ .

We define the OoO capability of an O3BNN 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 more OoO capability (discussed later). The pooling field is used when pooling is required after the convolution, where curve-groups belong 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 PE array, or are pruned.

**4.3.3 Data Feeding System.** DFS is designed to effectively feed a  $K \times K$  window for a curve-group from the input feature map, which is typically stored sequentially in  $[H, W, IC]$  order, while the  $K \times K$  window is from  $[H, W]$ . For efficiency, a simple segmented line-buffer design is proposed (see Figure 10): the entire input feature map flows along each segment of the line-buffer, with the  $K$  vertical segments extracting the required rows of the  $K \times K$  window and feeding into the PE array when a new curve-group is requested.

## 4.4 Design Extensions

In this section, we sketch extensions to O3BNN. First, we analyze how different OoO capability of O3BNN affects the performance of BNN inference. Second, we add a relaxing factor to the threshold and discuss how the relaxing factor affects the accuracy and performance.

**4.4.1 OoO capability.** Because of in-order commitment, when the Scoreboard has only one column of entries it can only track  $NOC$  curve-groups for the pixels with the same  $[W, H]$  at the same time. New curve-groups with new coordinators cannot be issued before the present curve-groups are completely evaluated, which may limit the OoO capability. If more hardware resources are available for the Scoreboard, we can track multiple  $NOC$  curve-groups (each per-column as shown in Figure 9), by initiating the coordinate field. This is a trade-off between hardware consumption and OoO capability: a larger Scoreboard provides higher OoO capability.

**4.4.2 Threshold Relaxing.** Until now all of the pruning designs are accuracy-lossless. Nevertheless, if accuracy can be compromised a little, we can gain more pruning benefits. The idea is to set a relaxing factor ( $\delta \in [0, 1]$ ) on the threshold. For condition 1, we relax  $T$  to a lower threshold,  $\delta \times T$ , so that condition 1 will be triggered earlier than it would have been. Similarly, for condition 2, we relax  $T$  to a higher threshold,  $(1 + (1 - \delta)) \times T$ . This is a trade-off between accuracy and pruning rate. When  $\delta$  is 1, threshold relaxing is not used and the pruning is lossless.

## 5 EVALUATION AND EXPERIMENTAL RESULTS

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

In our evaluation, we use Pytorch to train BNNs and augment Pytorch to profile the ideal pruning rates and measure accuracy. Accuracy is the inference accuracy on the testing set. For performance, hardware demand, and energy efficiency, we use 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 [2, 29]. The FPGA results are compared with two Intel CPUs (Xeon-E5 2640 [20] and Xeon-Phi 7210 [13]), two NVIDIA GPUs (Tesla-V100 and GTX-1080 [13]), and three FPGA systems: FINN [29], ReBNet [10], FP-BNN [20]. For networks, we use the well-known AlexNet and VGG-16 tested on the ImageNet dataset. We also use a VGG-like network for cifar-10, which is also widely used in the BNN literature [2, 13, 20, 29]. The structure configurations of these networks are listed in the first 3 rows of Table 1. Since FINN adjusts the structure of the VGG-like network (at the first row of Table 1), to make a fair comparison with FINN, we also test a VGG-like network with the same structure as the one used in FINN (noted as VGG-Like-FINN). Its structure is listed in the last row of Table 1.

### 5.1 Ideal Pruning Rate vs Network Accuracy

As mentioned in Section 3.2, there are 3 types of edge-pruning in our work: (1)&(2) “Condition 1” and “Condition 2” of threshold-based-edge-pruning; and (3) Pooling Pruning. Figure 11 shows the overall pruning rates of networks with the breakdown of the 3 types of pruning and network accuracy using different relaxing factors. For each network we use 5k random pictures from the test sets to profile the average pruning rates.

For lossless pruning, i.e.,  $\delta = 1$ , the top-1 accuracy is 88.5% and the pruning rate is 27% for VGG-Like. The top-5 accuracies are 72.7% and 75.5%, while the pruning rates are 19% and 42% for AlexNet and VGG-16. When the relaxing factor decreases, (1) the pruning rates increase almost linearly, especially for VGG-16 and

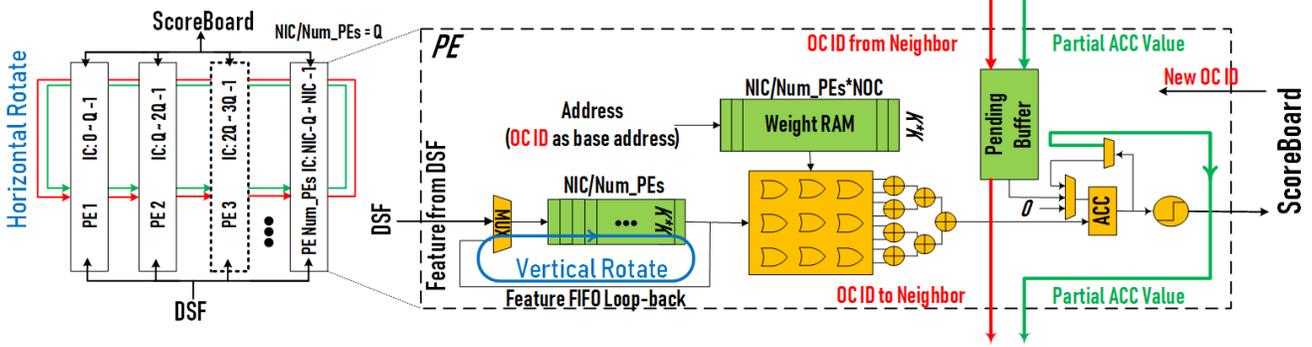


Figure 8: Architecture of PE array

Table 1: Structures of the Networks used to evaluate O3BNN. “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.

Network	Network Structure	Dataset	Input Image Size	Categories
VGG-like	(2x128C3)-MP2-(2x256C3)-MP2-(2x512C3)-MP2-(2x1024FC)	Cifar-10	32 × 32 × 3	10
AlexNet	(64C11/4)-MP3-(192C5)-MP3-(384C3)-(256C3)-MP3-(2x4096FC)	ImageNet	224 × 224 × 3	1000
VGGNet	(2x64C3)-MP2-(2x128C3)-MP2-(3x256C3)-MP2-(3x512C3)-MP2-(3x512C3)-MP2-(2x4096FC)	ImageNet	224 × 224 × 3	1000
VGG-like-FINN [29]	(2x64C3)-MP2-(2x128C3)-MP2-(2x256C3)-MP2-(2x512FC)	Cifar-10	32 × 32 × 3	10

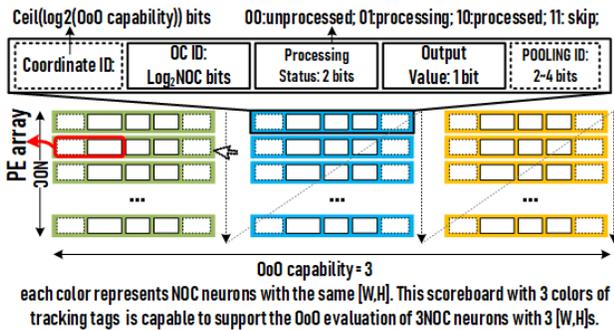


Figure 9: O3BNN Scoreboard

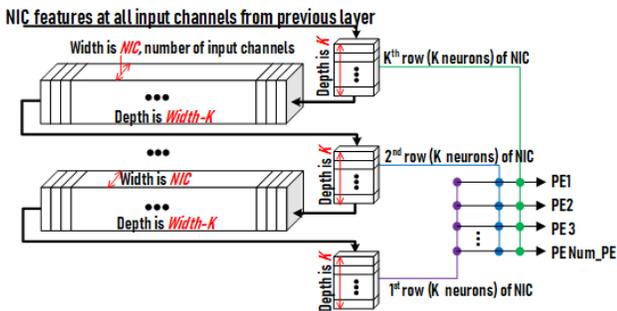


Figure 10: Architecture of DFS

factor reaches an inflection point after which it decreases rapidly. The reason is 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. AlexNet with large relaxing factors ( $\delta > 0.9$ ) is the outlier. When the relaxing factor used in AlexNet decreases from 1 to 0.9, neurons start to flip from 0 to 1, leading to increased 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  $3 \times 3$  max-pooling filter and  $11 \times 11$  CONV filter used in AlexNet. (2) When the relaxing factor is smaller than the inflection point, then the neurons’ values start to flip, i.e., 0 becomes 1 and 1 becomes 0, 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%, with only 3.3%, 0.9%, and 2.9% loss on accuracy, respectively. The networks of ImageNet are more sensitive to lowering the relaxing factors. The reason is that ImageNet has 1000 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.

VGG-Like; (2) the accuracy decreases slowly before the relaxing

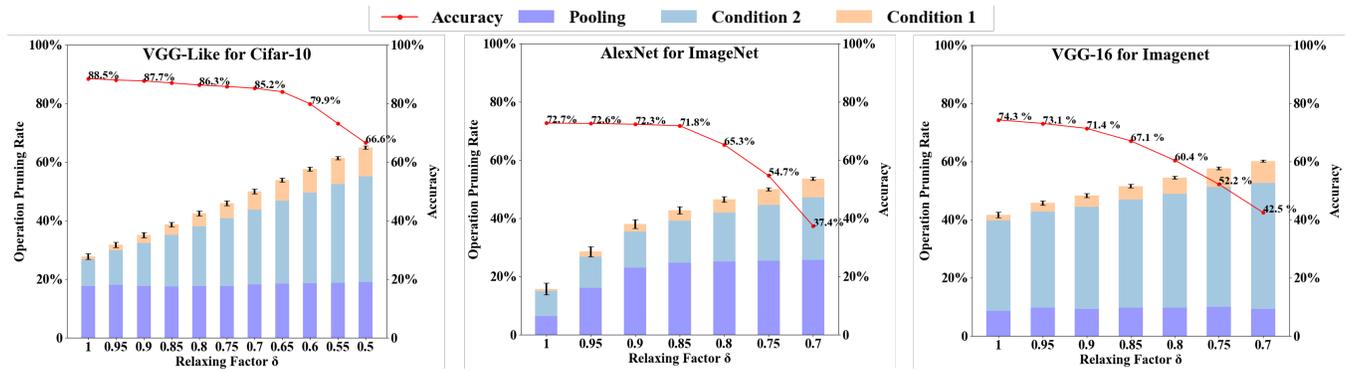


Figure 11: Pruning rates vs Accuracy trade-off with different relaxing factors. When relaxing factor is 1, the pruning is lossless.

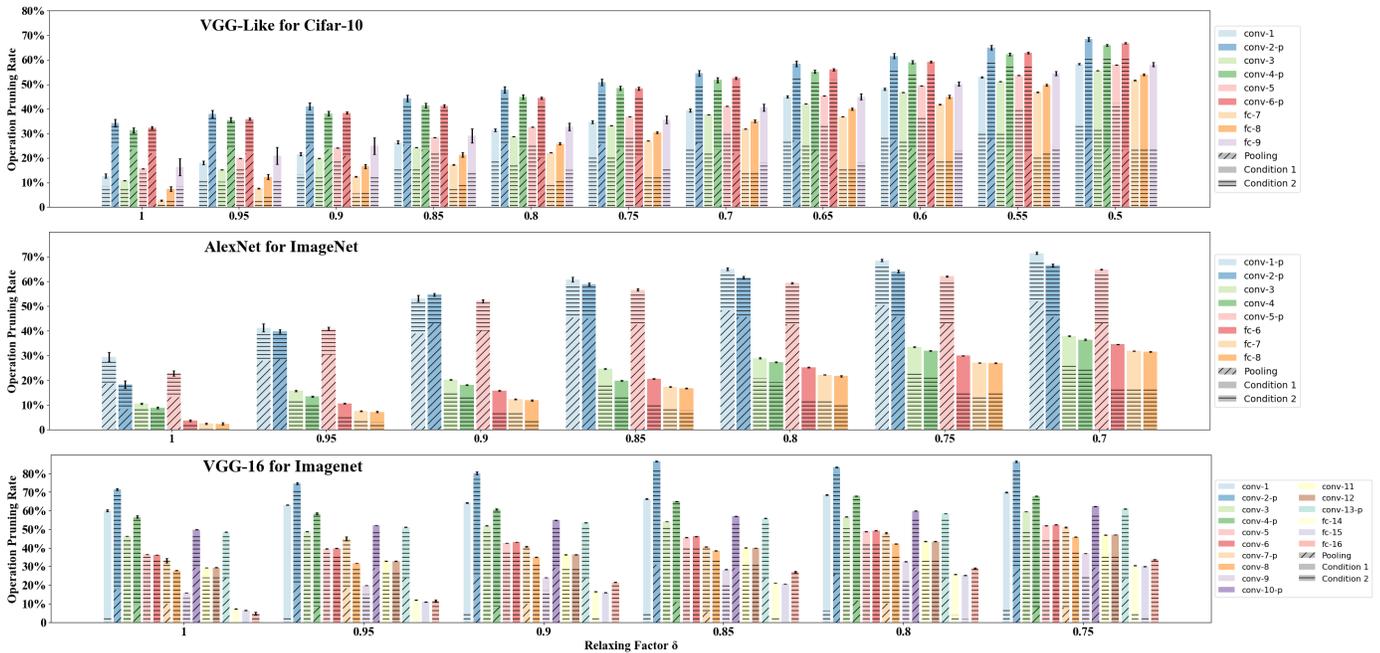


Figure 12: Pruning rates of different layers with different relaxing factors. When relaxing factor is 1, threshold relaxing is disabled and lossless pruning is used.

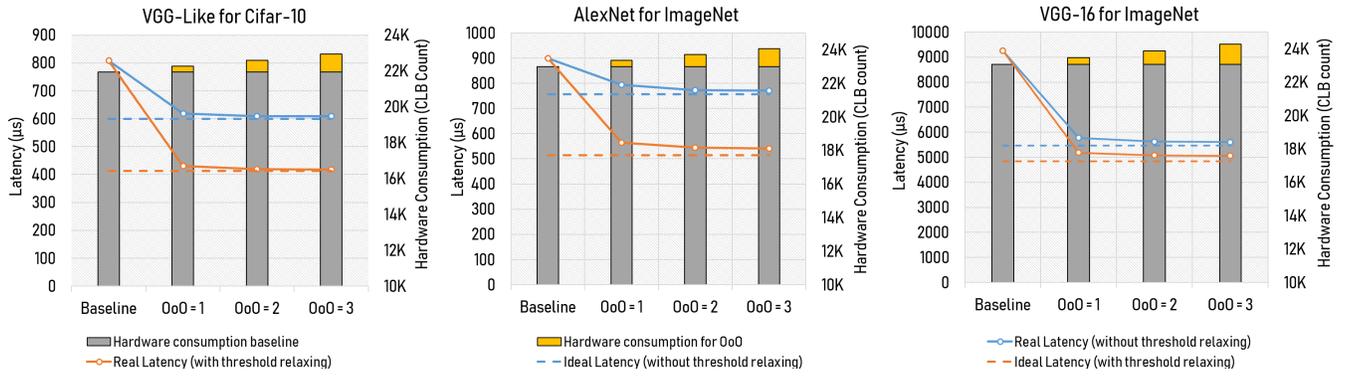
In our evaluation, to show the effect of different values of relaxing factors, all layers share the same relaxing factor. In practice, each layer can use a different relaxing factor. For the first two CONV layers and the last three FC layers, the relaxing factor can be relatively large because errors 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 can obtain higher pruning rates with less accuracy loss. Further analysis is left to future work.

Figure 12 is similar to Figure 11 but shows the pruning rates of each layer. It is observed that the pooling pruning is the most significant pruning type for all networks. 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, they increase much more rapidly than the CONV layers with the decrease in relaxing factor.

## 5.2 Hardware Demand versus Performance

By pruning the BNN network dynamically, O3BNN is expected to provide better performance than a traditional accelerator with no pruning. To evaluate the efficiency of O3BNN, we take the classic BNN inference implementation (described in Section 3.1) as the baseline and compare it to three O3BNN designs with different OoO capabilities. In the baseline design, Loops 1, 2, and 3 (in Figure 3)



**Figure 13: Performance and hardware consumption of O3BNNs with different OoO capabilities and with lossless (without threshold relaxing) or lossy (with threshold relaxing) pruning. O3BNNs are compared to baseline without pruning and ideal design with perfect pruning. The relaxing factors used in VGG-Like, AlexNet and VGG-16 are 0.7, 0.85 and 0.9 for lossy pruning.**

**Table 2: Latency, hardware demand and accuracy of baseline and O3BNN designs with OoO capability of 1, 2 and 3 and with lossless or lossy pruning. The relaxing factors used in VGG-Like, AlexNet and VGG-16 are 0.7, 0.85 and 0.9 respectively when lossy pruning is used.**

	OoO capability	VGG-Like			AlexNet			VGG-16		
		1	2	3	1	2	3	1	2	3
Without Threshold Relaxing	Latency( $\mu s$ )	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: VGG-Like: 0.7, AlexNet: 0.85, VGG-16: 0.9.	Latency( $\mu s$ )	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( $\mu s$ )	809			899			9251		
	Hardware Demand (CLB)	21930			23005			23056		
	Accuracy	88.5%			72.7%			74.3%		

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 [20, 23]. At each clock cycle, each PE calculates the value of one curve. Assuming there are  $NOC \times NIC$  PEs, at each cycle  $NOC$  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. Compared with the O3BNN 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). In addition, there is no Scoreboard in the baseline design, which uses static in-order scheduling.

Besides being compared with the no-pruning baseline implementation, the performance of O3BNNs is also compared to ideal performance, i.e., the performance of an ideal system which is able to exploit all pruning opportunities profiled in Section 5.1, and without any bubbles in the pipeline incurred by dynamic scheduling. The performance differences between the ideal performance

and the O3BNNs indicate the OoO processing efficiency. For each O3BNN design, 2 performance values are given: one for lossless pruning (threshold relaxing is not used and the relaxing factor is 1) and the other one for lossy pruning using the relaxing factor at the inflection point in Figure 13.

In Figure 13, the blue and orange lines indicate the latency using lossless and lossy pruning, respectively. Without any pruning, the inference latency of VGG-Like, AlexNet, and VGG-16 are 809, 899, and 9251  $\mu s$ , respectively. The hardware consumption is 21930, 23005, and 23056 Configurable Logic Block (CLBs). Using a O3BNN design whose OoO capability is 1, i.e., the Scoreboard can track the status of  $1 \times NOC$  curve-group at a time, the inference latency of these 3 networks are decreased to 619, 793, and 5779  $\mu s$  when using lossless pruning ( $\delta = 1$ ), and 430, 565, and 5186  $\mu s$  when relaxing factors at the inflection points are used. The hardware overheads are only 1.5%, 1.5%, and 1.8% compared with the baseline design. The performance of lossless and lossy O3BNNs with OoO capability of 1 are, on average, only 4.4%, and 6.5% lower than the ideal ones. The difference between ideal performance and the performance of lossy O3BNN is larger than the one between ideal and lossless O3BNN. The reason is that the pruning rates using lossy pruning are much larger than the ones using lossless pruning, requiring stronger OoO capability.

**Table 3: Cross-platform evaluation of Latency, Energy Efficiency and Accuracy. 4 Networks are used: VGG-like [3] and VGG-like-FINN [2] for Cifar-10 and AlexNet [19] and VGGNet-16 [26] for ImageNet. For lossy pruning,  $\delta$  are the relaxing factors at inflection points.**

	Existing works or self-implemented BNN inference as reference								
	CPU		GPU			FPGA			
Platform	Xeon E5-2640[20]	Phi 7210[13]	V100 (self-implemented)	GTX1080[13]	VCU108[10]	Stratix-V[20]	ZC706[29]		
Frequency (MHz)	2.5K	1.3K	1.37K			1.61K	200	200	
Network	VGG-Like	VGG-16	VGG-Like	AlexNet	VGG-16	AlexNet	AlexNet	VGG-Like-FINN	
Latency (us)	1.36E6	1.18E4	994	2226	1.29E4	1920	1160	283	
Energy (Img/kJ)	7.79	395	5543	2475	433	2.7E4	3.3E4	3.9E5	
Accuracy	86.31%	76.8%	89.9%	71.2%	76.8%	N/A	66.8%	80.1%	
O3BNN (OoO capability = 2)									
FPGA									
Platform	ZC706 (Stratix-V used in [20] and VCU108 used in [10] have hardware resources around 4× and 2.5× as much as ZC706 has)								
Frequency (MHz)	200								
Network	VGG-Like-FINN		VGG-Like		AlexNet		VGG-16		
Latency (us)	116 (lossy)	167 (lossless)	419 (lossy)	609 (lossless)	545 (lossy)	774 (lossless)	5080 (lossy)	5626 (lossless)	
Energy (Img/kJ)	9.58E5	6.65E5	2.65E5	1.82E5	2.04E5	1.44E5	2.19E4	1.97E4	
Accuracy	79.3%	82.6%	85.2%	88.5%	71.8%	72.7%	71.4%	74.3%	

When the OoO capability of O3BNN increases from 1 to 2, for lossless pruning, the latencies are reduced by 1.5%, 2.5%, and 2.6%, respectively for the three networks, and reach 609, 774, and 5626  $\mu$ s. The hardware demand is slightly increased, i.e., by 1.5%, 1.5%, and 1.7%. For lossy pruning with the relaxing factors at the inflection points, the latencies are reduced by 2.6%, 3.7%, and 2.1%, respectively and reach 419, 545, and 5080  $\mu$ s. The performance of O3BNN with OoO capability of 2 is only 5% lower than the ideal, on average. When the OoO capability is adjusted from 2 to 3, there is almost no additional performance improvement, but the hardware demand increases on average by 1.6%.

The latency, hardware demand, and accuracy of baseline and O3BNN-based BNN implementations are summarized in Table 2. As mentioned 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 our experimental results, the support of OoO processing of  $2 \times NOC$  curve groups is already enough for the unbalance of the edge pruning timing in BNNs. This result indicates that, if we retrain the network and push the thresholds to either 0 or the maximal accumulation value of a curve-group, then the pruning rates may increase while the pruning timing may become much more unbalanced. In this case, O3BNN architecture brings even more benefits.

### 5.3 Cross-platform Evaluation

In Table 3, O3BNN’s performance, energy efficiency, and accuracy (with lossless and lossy pruning) are compared with the existing and self-implemented systems using various CPUs, GPUs, and FPGAs to accelerate BNN inference. 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 Section 5.1 and 5.2, the OoO capability of O3BNNs is set as 2; in the case where threshold relaxing is used,

the relaxing factors in the inference of VGG-Like, VGG-Like-FINN, AlexNet and VGG-16 are 0.7, 0.7, 0.85 and 0.9 respectively.

Compared with FINN [2, 29], using the same network (i.e., VGG-Like-FINN) and FPGA board (i.e., ZC706), our lossless and lossy O3BNN demonstrate 167 & 116  $\mu$ s single-image inference latency for 1.7× & 2.4× speed-ups. Compared with FP-BNN [20] and ReBNet [10], our AlexNet inference latency using lossless and lossy pruning are 774 & 545  $\mu$ s, 1.50× & 2.13× over FP-BNN, and 2.48× & 3.52× over ReBNet. Note that the FPGA boards used in FP-BNN (i.e., Stratix-V) and ReBNet (i.e., VCU108) are actually high-performance FPGAs, which contain 4x and 2.5x the hardware resources as the one adopted for our evaluation (ZC706, an embedded FPGA). The energy efficiency of O3BNN also outperforms other FPGA work. Regarding other platforms, our inference latency with lossless and lossy pruning are 47.7% & 43.1% of the latency of Xeon-phi 7210 [13] and 43.6% & 39.4% of the latency of GXT 1080 [13], showing great advantage on performance. For energy-efficiency, the O3BNNs with lossless and lossy pruning are 50× & 55× higher than Xeon-phi 7210 and 45× & 51× higher than GXT 1080. The energy efficiency of baseline designs are not listed in Table 3. For VGG-Like-Finn, AlexNet and VGG-16, the baseline results are 5.07E5, 1.25E5, 1.22E5 Img/kJ respectively; In our paper, we use FPGAs as the evaluation platform. We believe the ratios of the overhead on hardware demands, the improvement of performance and energy efficiency of ASIC implementation are comparable to our FPGA results. The exact numbers depend on the choices of EDA tools and process technologies.

## 6 RELATED WORK

BNNs have been implemented variously [10, 13, 20, 23, 29, 32]. Because of the flexibility and direct bit-manipulation capability of FPGAs [7, 9, 25], most BNN implementations are FPGA-based [10, 20, 23, 29, 32]. We have already discussed FINN [29] in Section 2. In [32], Zhao, et.al., proposed the first high-level-synthesis-based BNN implementation on FPGAs. In [20], 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 [13] 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 [5] in Section 2. In [18], Li, et.al., proposed a new training method for BNNs in which a 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 the inference phase, their 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 [11, 22, 31], our design has three distinguishing aspects: (1) Run-time dynamic pruning for post-training network models; (2) Without compromising accuracy and no need to optimize or modify the training process; (3) 2D-rotative OoO-architecture to handle irregular parallelism from run-time dynamic pruning. Our future work includes applying O3BNN on general CNNs.

## 7 CONCLUSIONS

We propose O3BNN, an OoO high-performance BNN inference architecture with fine-grained and dynamic pruning. The contributions of O3BNN are two-fold. For algorithm, using O3BNN, the highly-condensed BNN model can be further shrunk significantly by dynamically pruning irregular redundant edges at all CONV, FC, and POOLING layers. For architecture, O3BNN is an out-of-order architecture which (1) checks the redundancy of edges at run-time and in a fine-grained-manner; (2) ceases edge evaluation in case the binary output of a neuron 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. 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%, 16%, and 42% of the operations for the three networks respectively, without any accuracy loss, leading to, at least, 1.7 $\times$ , 1.5 $\times$ , 2.1 $\times$  inference-speedup over state-of-the-art FPGA/GPU/CPU BNN implementations. With only 3.3%, 0.9% and 2.9% accuracy loss, the pruning rate increases to 49%, 43%, 48%, respectively, with, at least, 2.4 $\times$ , 2.1 $\times$ , and 2.3 $\times$  speedup. Our approach is inference runtime pruning, so no retrain or fine-tuning 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 NSF through Awards CNS-1405695 and CCF-1618303/7960; by the NIH through Award 1R41GM128533; by grants from Microsoft and Red Hat; and by

Intel through donated FPGAs, tools, and IP. This research was also partially funded by the Deep Learning for Scientific Discovery Investment under Pacific Northwest National Laboratory’s Laboratory Directed Research and Development Program. The evaluation platforms were supported by U.S. DOE Office of Science, Office of Advanced Scientific Computing Research, under award 66150: “CE-NATE - 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] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. 2016. Fused-layer CNN accelerators. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Press, 22.
- [2] Michaela Blott, Thomas B Preusser, Nicholas J Fraser, Giulio Gambardella, Kenneth O’Brien, Yaman Umuroglu, Miriam Leiser, and Kees Vissers. 2018. FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 11, 3 (2018), 16.
- [3] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*. 3123–3131.
- [4] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1. *arXiv preprint arXiv:1602.02830* (2016).
- [5] Tomoya Fujii, Shimpei Sato, and Hiroki Nakahara. 2018. A Threshold Neuron Pruning for a Binarized Deep Neural Network on an FPGA. *IEICE Transactions on Information and Systems* 101, 2 (2018), 376–386.
- [6] Tong Geng, Tianqi Wang, Ang Li, Xi Jin, and Martin Herbordt. 2019. A Scalable Framework for Acceleration of CNN Training on Deeply-Pipelined FPGA Clusters with Weight and Workload Balancing. *arXiv preprint arXiv:1901.01007* (2019).
- [7] Tong Geng, Tianqi Wang, Ahmed Sanaullah, Chen Yang, Rushi Patel, and Martin Herbordt. 2018. A Framework for Acceleration of CNN Training on Deeply-Pipelined FPGA Clusters with Work and Weight Load Balancing. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 394–3944.
- [8] Tong Geng, Tianqi Wang, Ahmed Sanaullah, Chen Yang, R Xuy, Rushi Patel, and Martin Herbordt. 2018. FPDeep: Acceleration and load balancing of CNN training on FPGA clusters. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*.
- [9] Alan D George, Martin C Herbordt, Herman Lam, Abhijeet G Lawande, Jiayi Sheng, and Chen Yang. 2016. Novo-G#: Large-scale reconfigurable computing with direct and programmable interconnects. In *2016 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 1–7.
- [10] Mohammad Ghasemzadeh, Mohammad Samragh, and Farinaz Koushanfar. 2018. ReBNet: Residual Binarized Neural Network. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 57–64.
- [11] Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*. 1389–1397.
- [12] John L Hennessy and David A Patterson. 2011. *Computer architecture: a quantitative approach*. Elsevier.
- [13] Yuwei Hu, Jidong Zhai, Dinghua Li, Yifan Gong, Yuhao Zhu, Wei Liu, Lei Su, and Jiangming Jin. 2018. BitFlow: Exploiting Vector Parallelism for Binary Neural Networks on CPU. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 244–253.
- [14] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. In *Advances in neural information processing systems*. 4107–4115.
- [15] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research* 18, 1 (2017), 6869–6898.
- [16] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 2013. 3D convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence* 35, 1 (2013), 221–231.
- [17] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. 2014. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 1725–1732.
- [18] Soroosh Khoram and Jing Li. 2018. Adaptive Quantization of Neural Networks. (2018).

- [19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.
- [20] Shuang Liang, Shouyi Yin, Leibo Liu, Wayne Luk, and Shaojun Wei. 2018. FP-BNN: Binarized neural network on FPGA. *Neurocomputing* 275 (2018), 1072–1086.
- [21] Liqiang Lu, Yun Liang, Qingcheng Xiao, and Shengen Yan. 2017. Evaluating fast algorithms for convolutional neural networks on fpgas. In *Field-Programmable Custom Computing Machines (FCCM), 2017 IEEE 25th Annual International Symposium on*. IEEE, 101–108.
- [22] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2016. Pruning convolutional neural networks for resource efficient transfer learning. *arXiv preprint arXiv:1611.06440* 3 (2016).
- [23] Eriko Nurvitadhi, David Sheffield, Jaewoong Sim, Asit Mishra, Ganesh Venkatesh, and Debbie Marr. 2016. Accelerating binarized neural networks: comparison of FPGA, CPU, GPU, and ASIC. In *Field-Programmable Technology (FPT), 2016 International Conference on*. IEEE, 77–84.
- [24] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*. Springer, 525–542.
- [25] Jiayi Sheng, Chen Yang, and Martin C Herbordt. 2018. High performance communication on reconfigurable clusters. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 219–2194.
- [26] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
- [27] Wei Tang, Gang Hua, and Liang Wang. 2017. How to train a compact binary neural network with high accuracy?. In *AAAI*. 2625–2631.
- [28] Robert M Tomasulo. 1967. An efficient algorithm for exploiting multiple arithmetic units. *IBM Journal of research and Development* 11, 1 (1967), 25–33.
- [29] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. 2017. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 65–74.
- [30] Xuechao Wei, Cody Hao Yu, Peng Zhang, Youxiang Chen, Yuxin Wang, Han Hu, Yun Liang, and Jason Cong. 2017. Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs. In *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, 29.
- [31] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. 2017. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5687–5695.
- [32] Ritchie Zhao, Weinan Song, Wentao Zhang, Tianwei Xing, Jeng-Hau Lin, Mani Srivastava, Rajesh Gupta, and Zhiru Zhang. 2017. Accelerating binarized convolutional neural networks with software-programmable fpgas. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 15–24.
- [33] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2016. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160* (2016).