MST-compression: Compressing and Accelerating Binary Neural Networks with Minimum Spanning Tree

Quang Hieu Vo, Linh-Tam Tran, Sung-Ho Bae, Lok-Won Kim, Choong Seon Hong*
Department of Computer Science and Engineering, Kyung Hee University
{2019310178, tamlt, shbae, lwk, cshong}@khu.ac.kr

Abstract

Binary neural networks (BNNs) have been widely adopted to reduce the computational cost and memory storage on edge-computing devices by using one-bit representation for activations and weights. However, as neural networks become wider/deeper to improve accuracy and meet practical requirements, the computational burden remains a significant challenge even on the binary version. To address these issues, this paper proposes a novel method called Minimum Spanning Tree (MST) compression that learns to compress and accelerate BNNs. The proposed architecture leverages an observation from previous works that an output channel in a binary convolution can be computed using another output channel and XNOR *operations with weights that differ from the weights of the reused channel. We first construct a fully connected graph with vertices corresponding to output channels, where the distance between two vertices is the number of different values between the weight sets used for these outputs. Then, the MST of the graph with the minimum depth is proposed to reorder output calculations, aiming to reduce computational cost and latency. Moreover, we propose a new learning algorithm to reduce the total MST distance during training. Experimental results on benchmark models demonstrate that our method achieves significant compression ratios with negligible accuracy drops, making it a promising approach for resource-constrained edge-computing devices.*

1. Introduction

Deep Neural Networks (DNNs) have been widely applied in many artificial intelligence applications, especially vision tasks with high accuracy [27, 14]. However, the cost of computation and massive storage burden is significantly challenging to deploy DNNs on embedded systems such as mobile devices and other resource-constrained platforms. Many approaches have been proposed and demonstrated their effectiveness in reducing energy and resources while maintaining the deep models' accuracy, including pruning [12], quantization [7], distillation [15], and efficient hardware implementation [6]. Among these methods, quantization with less bit-width for parameter and activation representation is widely used due to its great benefits and possibility in most practical applications.

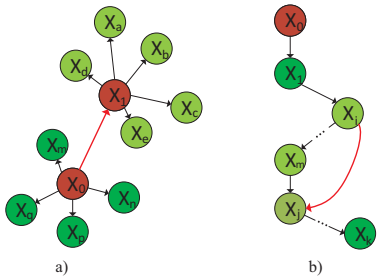


Figure 1. Illustration of kernel compression for BNNs with the K-mean method (a) and the shortest Hamiltonian path (b), in which adding red connections can further reduce the computational cost for both of them.

Binarized neural network is a particular form of the quantization method in which weight values and activations are converted to 1-bit values. Accordingly, multiplications and accumulation operations can be replaced by XNOR and Popcount operations [7], respectively. In addition, batch normalization is simplified to a threshold comparison [31], while the pooling can be performed with OR operations [30]. Consequently, the compression ratio on memory storage and computational cost is significantly improved, leading to remarkable performance acceleration. Nevertheless, the accuracy degradation is the trade-off of minimizing the bit-width as BNN. Thus, most previous works focus on reducing this accuracy gap [25, 4, 33, 22, 20, 33, 23].

Meanwhile, compressing BNNs has not received much attention, with only a few prominent methods [32, 9, 31, 17], where the kernel compression [9, 31] gives impressive results with roughly 50% resources reduction. In particular, given a binary convolution, inspired by an observation that an output channel can be calculated using another out-

*Corresponding Author

put channel and outputs of XNOR operations using weights that respectively differ from the reused channel's weights [9], authors in [31, 9] proposed to construct a fully connected graph, in which each vertex corresponds to an output channel, and distance between two vertices is the number of different values between two weight sets used for the two output channels. Then, based on the graph, they use the K-mean cluster and shortest Hamiltonian path to reorder the convolution output calculation, aiming to reduce the number of XNOR operations. However, these approaches have yet to fully minimize the computational cost as output channels can use less computation. Indeed, Figure 1 (a) illustrates an example of the K-mean method, where two vertices are considered centers of two groups. Two output channels corresponding to centers are fully calculated[1], while other vertices reuse their centers for calculation. However, adding a connection between the two centers would enable one to utilize the rest with less computation cost. Additionally, in the shortest Hamiltonian path shown in Figure 1 (b), there may exist a connection to a specific vertex that is shorter than the connection from the preceding vertex in the path, leading to a lower required number of operations for this vertex. Furthermore, time complexity poses a challenge in these methods [1, 13], resulting in longer exploration times.

To more effectively minimize the number of XNOR operations with the mentioned observation, in all connections to a specific vertex, the minimum connection must be selected to minimize the computation cost for this vertex. In addition, all of these selected connections must be included in a subgraph that visits all vertices exactly once to ensure that only one output channel is fully calculated. As a result, a MST is the only structure that can fulfill these requirements. Therefore, this paper leverages the MST to reorder binary convolution output calculations. Figure 2 shows a simple example of reordering calculation on a convolution. In this example, only the output channel 4 is fully calculated with $C_{in} \times M \times M$ bit-weight values. In contrast, other channels are calculated using output channel 4, following the MST direction.

On the other hand, to maximize the advantages of the MST, we further minimize the MST distance of all convolution layers with a new learning algorithm right during the training stage. Besides, we propose a hardware accelerator for BNNs applying the MST compression to demonstrate the feasibility and effectiveness of the method related to hardware resources. To the best of our knowledge, this method gives the highest compression ratio with implementation from learning for compression to acceleration. In summary, the following are contributions of this paper:

- We introduce and analyze the effectiveness of the MST in reducing the computational cost for the inference on a binary convolution layer.
- We propose a training algorithm that can reduce the MST distance and depth right during the training process, which consequently maximizes the compression ratio for inference implementation.
- We provide the corresponding hardware acceleration for the proposed method with high throughput and better resource efficiency, compared to related works [3, 30, 31, 10].

[1] calculated with $C_{in} \times M \times M$ XNORs, denoted in Figure 2

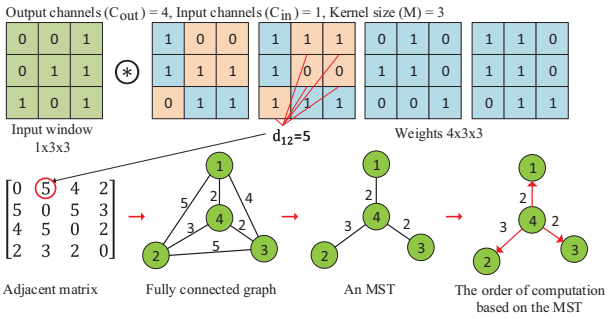


Figure 2. The process of arranging the order of computation on a binary convolution layer, in which output channel 4 is fully calculated first. Then, the output channel 4 is used to calculate channel 1, channel 2 and channel 3.

The experiments are performed for BNNs including VGG-small [5], ResNet-18/20 [14] on CIFAR-10 dataset [18], and ResNet-18/34 [14] on ImageNet [26]. The results show that the proposed approach gives a higher compression ratio than the previous works [32]. Compared to the baseline [33], our method reduces up to $13.5\times$ the convolution parameters and $5.51\times$ bit-wise operations on the same model with an acceptable accuracy degradation. Regarding hardware acceleration, we conduct the experiments for a BNN with the same structure as the baseline [31] and apply the proposed approach. Compared to the baseline [31], hardware deployments demonstrate that BNNs with our method save $1.8\times$ LUTs, and $1.81\times$ area efficiency while maintaining the acceleration speed and accuracy.

2. Related Work

Training neural networks with binary weights and activations were first introduced by Coubariaux *et al*. [7] and then rapidly gained attention from the community due to the high compression ratio. However, accuracy drop is a critical problem of this direction, while compression is still highly demanded. Various techniques are proposed to further compress and improve accuracy based on the binary property.

For accuracy improvement, Rastegari *et al*. [25] proposed adding a scaling factor to the convolution to reduce the quantization error. To generalize this idea aiming to enhance accuracy, the authors in [4] extended this factor dimension and enabled its training option as a learnable parameter. Besides, instead of using the Straight Through Estimation (STE) method [2], Liu *et al*. [23], and Lin *et al*. [20] proposed using a piece-wise linear and training-aware approximation function, respectively, to approximate the sign function gradient better. Meanwhile, some previous works introduced both new optimization techniques and corresponding BNN to shrink the accuracy gap entirely [22, 21, 28, 24]. For example, based on Bi-real Net in [23], authors in [22] proposed new sign and activation functions to shift and reshape activation distribution with a new baseline neural network model that can boost the performance. Other works look at the binary characteristics from deeper perspectives to limit information loss [20, 28, 33]. For example, KL divergence is employed in [28] to minimize the difference between the distribution of binary and real-weight counterparts. Xu *et al*. [33] observed the low probability of changing large full-precision weight when binarizing and proposed the ReCu function to revive the death weights, leading to lower quantization error.

Binary model compression has received less attention as only a few techniques have been proposed so far. Specifically, Lee *et al*. [19] proposed an encryption algorithm to compress binary weights with lower than 1-bit per weight, and XOR-gate networks are used to decrypt the inference task. Similarly, Sub-bit Neural Networks (SNN) [32] proposed using lower than 9 bits for each 3×3 binary kernel based on the scattered 9-bit kernel distribution. Other proposed compression methods provided only hardware techniques for inference implementation. Authors in [31, 9] proposed BNN hardware architectures using weight/input reuse with different calculation orders for convolution to reduce hardware overhead. Kim *et al*. [17] introduces kernel decomposition that can reduce the computational cost by sharing one base output for all channels.

3. Background

This section briefly describes how to construct a binary convolution layer and the related optimization methods. For a convolution layer with C_{in} input channels, C_{out} output channels, and kernel size M, weights and activations are denoted by $\mathcal{W}^r \in \mathbb{R}^n$ and $A^r \in \mathbb{R}^m$, where $n = C_{out} \times C_{in} \times M \times M$ and $m = C_{in} \times W_{in} \times H_{in}$. $W_{in} \times H_{in}$ represents input feature map size. To simplify the computation and reduce memory storage, in a binary convolution layer, the binarization of activations $\mathcal{A}^b \in \{\pm 1\}^m$ and weights $\mathcal{W}^b \in \{\pm 1\}^n$ is acquired by the following sign function [25],

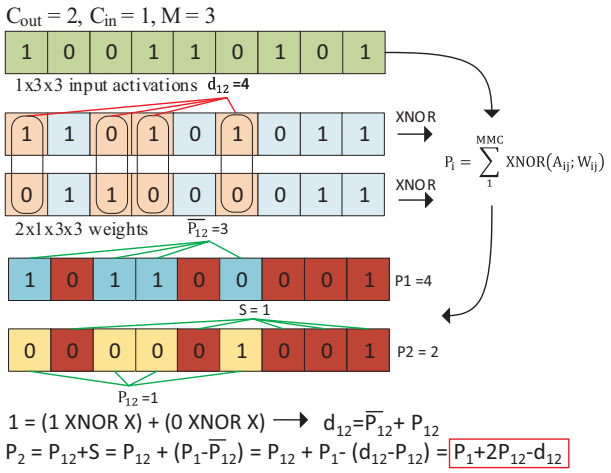


Figure 3. Weight-reuse method description. Here, the red line is counting the number of elements, the green line is summing all elements. P_{12} is $\sum_{j=1}^{d_{12}} \text{XNOR}(\mathcal{A}_{2j}, \mathcal{W}_{2j})$, and Y_2 is $2(P_1 - d_{12} + 2P_{12}) - C_{in} \times M \times M$.

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

If -1 is substituted by 0 to represent a negative value for $\mathcal{A}^b$ and $\mathcal{W}^b$, the output convolution operation of the i^{th} channel now is reformulated as in Eq. (2) [31].

$$Y_i = (2 \sum_{j=1}^{C_{in}MM} \text{XNOR}(\mathcal{A}^b_{ij}, \mathcal{W}^b_{ij}) - C_{in} \times M \times M) \odot \alpha, \tag{2}$$

where $\sum_{1}^{C_{in} \times M \times M} \text{XNOR}(\mathcal{A}^b_{ij}, \mathcal{W}^b_{ij})$ is the number of set-bits (Popcount) of the output channel i^{th}, $\odot$ is the element-wise multiplication, and $C_{in} \times M \times M$ is the number of bit-wise XNOR operations used to calculate one output channel. The scaling factor α is added as a learnable parameter to lessen quantization error [25]. For backward propagation, following [23], the STE [2] method is used for the approximation gradient on weights, while the piece-wise polynomial function [23] is used for gradient approximation on activations.

To further compress the BNN models, in this paper, we apply the weight reuse method [31] for the inference task. Accordingly, given the l^{th} binary convolution, as shown in Figure 3, we denote P_i and P_j as the output of the Popcount operations of the output channels i^{th} and j^{th}, respectively. All binary weights of the layer are divided into C_{out} weight sets $w_b^{li} \in \{\pm 1\}^{C_{in} \times M \times M}$, where $i = 1, 2, ..., C_{out}$ and each weight set is used to calculate for an output channel. d_{ij} is the number of weight bits in the w_b^{li} that differ from the w_b^{lj}, compared one-to-one respectively. P_{ij} is the sum of XNOR operations with the

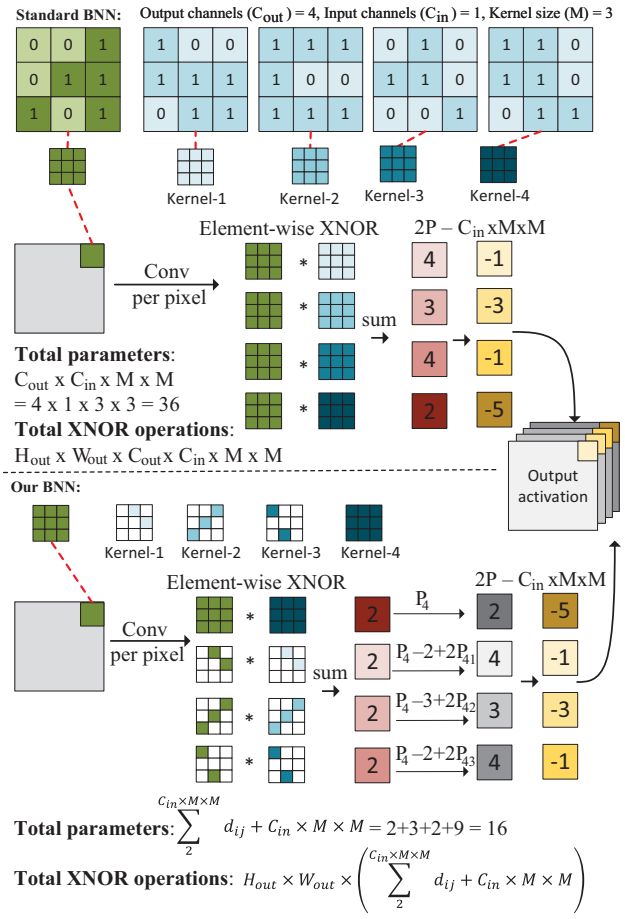


Figure 4. Comparison between the computation process of a standard binary convolution and our method in BNN acceleration. Instead of fully computing all $C_{in} \times M \times M$ XNOR operations for each layer, our method provides an optimal order of the calculation using the MST to minimize the number of XNOR operations and total parameters.

weights of the channel j^{th} that differ from the i^{th} counterpart. To calculate the output channel j^{th} from the output channel i^{th}, we use Eq. (3).

$$Y_j = 2(P_i - d_{ij} + 2P_{ij}) - C_{in} \times M \times M. \quad (3)$$

4. Methodology

4.1. Minimum Spanning Tree for BNN Acceleration

For a binary convolution layer, we construct a fully connected graph in which each vertex corresponds to an output channel. The distance between two vertices i and j is d_{ij}. As shown in Eq. (3), an output channel j can be calculated via the Popcount of an output channel i and the Popcount of d_{ij} XNOR operations. That means the number of XNOR operations executed for the output channel j is the distance d_{ij}. Based on this observation, to minimize the number of XNOR operations used for all output channels, we need a sub-graph that includes all vertices with the minimum total edge distances, which is named an MST [29]. Returning to the example in Figure 2, Figure 4 illustrates the computation process of this convolution example with the standard and proposed approach. Specifically, in the standard direction, each output channel needs 9 XNOR operations, while in our direction, only the 4^{th} output channel is fully computed by 9 XNOR operations, and the 1^{st}, 2^{nd}, 3^{rd} output channels reuse the output channel 4^{th} with 2, 3, 2 XNOR operations, respectively. Accordingly, the compression ratio for this example is $(2+3+2+3\times3)/(4\times1\times3\times3) = 0.44$. In general, we can reduce the computational cost for a convolution layer with the following ratio:

$$\mathcal{R} = \frac{\sum_{j\neq root}^{C_{out}} d_{ij} + C_{in} \times M \times M}{C_{out} \times C_{in} \times M \times M}, \quad (4)$$

where i is the parent of the vertex j. If j is the root of the MST, this output uses $C_{in} \times M \times M$ binary weight values.

Regarding parameters, because the streaming architecture described in Sec. 4.3 is fully implemented with all layers on the FPGA platform, weights' locations can be converted to connections from inputs to the correct XNORs and corresponding weight values at the circuit level. Thus, our proposed method counts only the parameters we need for XNOR logic gates, as shown in Figure 4.

Number of bit-wise operations. Compared with the previous approaches [31, 9], in a binary convolution layer, the proposed method can better reduce the number of XNORs and parameters. Firstly, as explained in the previous section, adding connections among centers leads to less computation for the K-mean approach [31] as in Eq. (5).

$$\sum_{j} d_{ij} + C_{in} \times M \times M < \sum_{j\notin C} d_{ij} + R \times C_{in} \times M \times M, \quad (5)$$

where C includes all centers, R is the number of centers. Furthermore, adding connections among centers also creates a spanning tree, whose computational cost is always equal to or higher than using an MST. Secondly, in [9], the authors proposed using the shortest Hamiltonian path to arrange the computation order. Because this path visits all vertices exactly once, the shortest Hamiltonian path is also a spanning tree. Therefore, the number of XNOR bit-wise operations using an MST method is equal to or lower than the shortest Hamiltonian path.

Exploration time. The MST is faster and less complex for finding the computation order, compared to previous work [9, 31]. Indeed, the MST is found by Prim's algorithm with the complexity of $O(V^2)$ [29], where V is the number of vertices. In contrast, the K-mean algorithm requires $O(V^2G)$ [13] time complexity, where G denotes the number of groups. Since the optimal number of groups varies

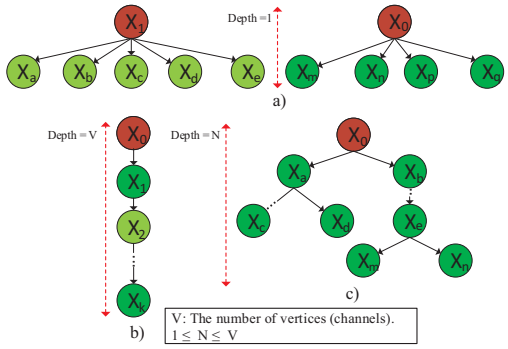


Figure 5. a) The computation tree's depth using the K-mean cluster. b) The computation tree's depth using the shortest Hamiltonian path. c) The computation tree's depth using the MST.

L	Hamiltonian		K-mean		MST	
	Bit-Ops	Time	Bit-Ops	Time	Bit-Ops	Time
1	1.08 M	31.3 s	1.33 M	0.090 s	1.07 M	0.021 s
2	1.09 M	32.9 s	1.30 M	0.075 s	1.06 M	0.021 s
3	1.07 M	32.6 s	1.30 M	0.085 s	1.05 M	0.023 s
4	1.10 M	32.7 s	1.33 M	0.076 s	1.08 M	0.021 s
5	1.11 M	32.5 s	1.32 M	0.115 s	1.09 M	0.020 s
6	1.09 M	32.5 s	1.33 M	0.086 s	1.08 M	0.021 s

Table 1. Bit-Ops and running time w.r.t. different approaches for reordering calculation on the first 6 convolution layers of ResNet-20 with 16 output channels.

across convolutions, extra time is needed to determine this number for each case. Meanwhile, the shortest Hamiltonian path has a much higher time complexity of $O(V^2 2^V)$ [1], making it challenging to apply in practical settings. To estimate these approaches for practical operation, we perform experiments for the first 6 binary convolution layers of ResNet-20 [14] with the weight data trained by [33], while experiments on deeper layers with a larger number of output channels are infeasible even uncompleted for the K-mean and the shortest Hamiltonian path approach due to their longer running times. According to Table 1, the MST approach gives the lowest bit-Ops and exploration time.

Computation latency. This is one of the factors affecting the resource overhead. This paper focuses on improving the architecture in [31], a streaming acceleration for BNNs. In doing so, the deeper the computation tree, the longer latency between the time output of the first and the last channel comes out. Consequently, the number of added registers (Flip-flops) for synchronization tends to be equal or more for the deeper tree because all output channels must be available at the same clock for the next convolution operation. The exact number of added registers depends on the number of adders executed in a clock period, which is affected by the required frequency, design complexity, and hardware platform. Thus, to evaluate the comprehensive effects of the depth with practical implementations, the experiments related to this comparison are provided in Sec. 5.

Figure 5 visually describes the depth of the MST, K-mean cluster, and the shortest Hamiltonian path approach. Accordingly, the depths of the K-mean method and the shortest Hamiltonian path are always 1 and V, respectively. Meanwhile, the MST's depth can range from 1 to V, depending on the graph structure. Therefore, the proposed method is always better than the shortest Hamiltonian path in all aspects (number of operations, latency, running time). Meanwhile, the K-mean method gives a better latency but much worse for the rest. After exploring an MST for the post-training stage to limit the impact of the tree depth on latency and resources, we continue finding a new root that makes the depth of the new tree minimum.

4.2. Learning Optimization

According to the observation mentioned earlier, to enhance the benefits of the MST approach, we propose a simple learning method that can further reduce the MST depth and total MST distance over a BNN model. Specifically, to make the training converged, instead of directly optimizing the MST, we cluster weight sets (w_b^{li}) to different groups and try to reduce the distance among weight sets in each group by optimizing the distances from all weight sets to fixed centers. In doing so, given the l^{th} binary convolution layer of a BNN model, we first randomly sample a binary center subset $\mathbb{C}_l \subset \{\pm 1\}^{C_{in} \times M \times M}$ and $|\mathbb{C}_l| = N_l$. Then, we explore the nearest center in $\mathbb{C}_l$ for every w_b^{li} and save it into a dictionary $\mathcal{C}$ using Eq. (6).

$$\mathcal{C}(\mathrm{w}_b^{li}) = \underset{x}{\operatorname{argmin}} \left(\sum_{i=1}^{C_{in} \times M \times M} \|x - \mathrm{w}_b^{li}\| \right); x \in \mathbb{C}_l. \quad (6)$$

On the other hand, the total distance from every weight sets to its center is added to the loss function and compels the training to minimize both the cross-entropy loss and the distance among weight sets in a group (distance loss). To strike a balance between optimizing the original and distance loss, we adjust a hyper-parameter λ to achieve an optimal ratio for different BNN models. Besides, a γ parameter serves as a hyper-parameter to control the priority of the original and additional loss during the training. This parameter varies following a specific scheduler during the training process. Particularly, the algorithm prioritizes reducing the MST distance and depth during the initial training phase and gradually shifting its focus toward accuracy at the end of the process. The new loss function is shown in Eq. (7), where $\mathcal{L}_0(\mathrm{input}, \mathrm{w})$ is the original loss function. Forward processes are summarized in Algorithm 1.

$$\mathcal{L}(\mathrm{input}, \mathrm{w}) = \mathcal{L}_0(\mathrm{input}, \mathrm{w}) + \lambda\gamma \sum_i \|\mathcal{C}(\mathrm{w}_b^i) - \mathrm{w}_b^i\|^2, \quad (7)$$

It is noticeable that we use pre-trained BNN models for the learning process to avoid diverging during distance op-

Algorithm 1 Forward propagation for the training process.

Require: Input data, pre-train full-precision weight (w), threshold (θ), learning rate (η), MST scaling factor (λ).

Load pre-train weight (w)
for epoch = $1 \rightarrow E$ **do**
  $\mathcal{L}_{mst} = 0$
  for l = 1 to L **do**
    $\mathrm{w}_b^l = \mathrm{Sign}(\mathrm{w}_l)$ [33]
    $\mathrm{a}_b^{l-1} = \mathrm{Sign}(\mathrm{a}_{l-1})$
    for i = 1 to C_{out} **do**
      Update centers for w_b^l:
      $\mathcal{C}(\mathrm{w}_b^{li}) = \underset{x}{\mathrm{argmin}}(\sum_{i=1}^{C_{in}\times M\times M} \|x - \mathrm{w}_b^{li}\|) \, ; x \in \mathbb{C}_l$
      $\mathcal{L}_{MST} = \mathcal{L}_{MST} + \sum \|\mathcal{C}(\mathrm{w}_b^{li}) - \mathrm{w}_b^{li}\|^2$
    end for
    $\mathrm{a}_l = (\mathrm{w}_b^l \circledast \mathrm{a}_b^{l-1}) \odot \alpha_l$
  end for
  Compute total loss: $\mathcal{L} = \mathcal{L}_0 + \lambda * \gamma * \mathcal{L}_{MST}$
end for

timization time. In this paper, results from ReCu [33] are used and considered the baseline for comparison.

4.3. Streaming Acceleration

Inspired by the streaming architecture in [31], this paper proposes an entire BNN high-speed architecture for BNN models applying our compression method. Specifically, the hardware architecture of a binary convolution layer is shown in Figure 6, in which the process is divided into two steps: 1) loading the input feature map to a line buffer and 2) executing XNOR + Popcount operation.

Loading input. Given a binary convolution layer with $M = 3$, padding size $= 1$, firstly, C_{in} input feature maps are gradually loaded into C_{in} line buffers with the size $2 \times W_{in} + 3$. When $W_{in} + 2$ registers in each line buffer are filled, $C_{in} \times 3 \times 3$ input pixels are selected from the line buffers for the convolution (XNOR + Popcount) operation. In standard calculation approach, the number of XNOR operations used for a binary convolution is $C_{out} \times C_{in} \times M \times M$. These operations can be separated into C_{out} sub-operations corresponding to C_{out} output channels; each sub-operation includes $C_{in} \times M \times M$ XNOR bit-operations and requires $C_{in} \times M \times M$ corresponding input pixels and weight values. In this proposed architecture, the difference is that the number of inputs for a particular sub-operation is lower than $C_{in} \times M \times M$ and equal to the distance between the corresponding vertex and parent vertex as in the constructed graph, except the sub-operation for the root vertex requiring full calculation with $C_{in} \times M \times M$ XNORs.

Perform convolution. This step executes the convolution operation with the order of calculation following the MST. As shown in Figure 6, the output of a specific PE (red arrow) is reused for the next PE to reduce computational cost. The PE for the output channel i^{th} needs d_{ij} weight values and executes d_{ij} XNOR operations with the corresponding Popcount operation, where j is the parent

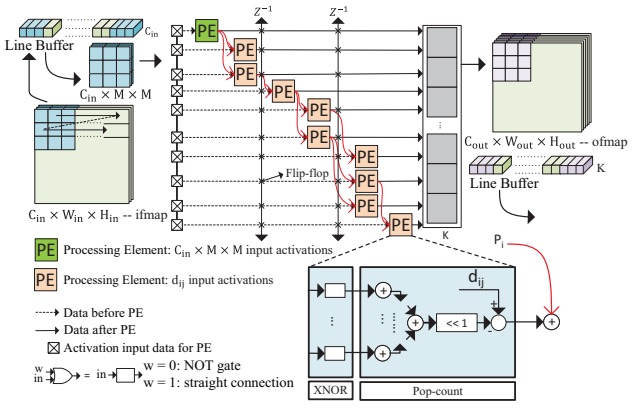


Figure 6. High-speed hardware accelerator for a binary convolution layer, where the XNORs are simplified with NOT or simple straight connection, while computation on each PE corresponding to each output channel is executed gradually with the MST.

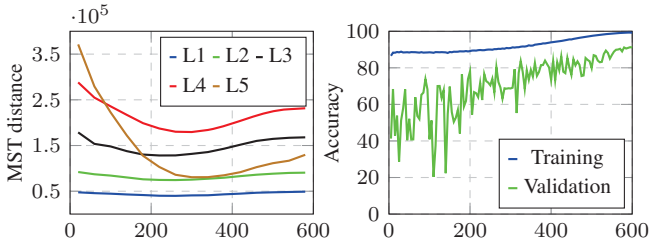
vertex of the vertex i^{th}. Besides, to eliminate memory for convolution weight storage, XNOR operations are simplified to NOT gate or straight connection. In terms of timing violations, we pipeline the Popcount operation by adding register lines among the adder tree to reduce computation complexity in a clock cycle. The exact number of register lines depends on the MST depth, frequency and hardware platform. Thus, depending on practical implementations, we adjust this number to meet the timing requirements.

5. Experiments

In this section, we conduct experiments including software and hardware implementation to show the effectiveness of the proposed method for BNN compression. Training experiments are executed with the popular datasets CIFAR-10 and ImageNet via Pytorch, while hardware implementation is performed on the Xilinx FPGA platforms.

5.1. Learning and hardware implementation

Training implementation: Because our proposed method is independent of other BNN accuracy optimization methods, in this paper, we select training results from the ReCU approach in [33] as a case study for further compressing. In doing so, VGG-small, ResNet-18/20 models are used to evaluate on CIFAR-10 dataset, while ResNet-18/34 are used for ImageNet. Regarding training in detail, we keep the configuration as in [33], except for the learning rate of 0.1. In addition, the λ value is adjusted depending on the training models, datasets, and focused objectives. Meanwhile, the γ parameter is scheduled following the learning rate. In this paper, when training with CIFAR-10, we select 1e-6 for ResNet-18, 5e-6 for ResNet-20, and 4e-6 for VGG-small. Meanwhile, for ImageNet, we select 1e-6 for ResNet-18 and 1e3 for ResNet-34.



a) MST distances of each layer (L). b) Accuracy curves.

Figure 7. MST distance and top-1 % accuracy during the training process, where the Binary VGG-small model and CIFAR-10 dataset is used for the training experiment with 600 epochs.

λ	MST-depth	#Params (Mbit)	#Bit-Ops (GOps)	Top-1 Acc. mean $\pm$ std (%)
1e-7	97.3	1.391	0.217	92.17 $\pm$ 0.07
5e-7	58.3	0.998	0.184	92.09 $\pm$ 0.06
1e-6	54.0	0.864	0.165	91.99 $\pm$ 0.07
5e-6	44.0	0.532	0.115	91.14 $\pm$ 0.08
1e-5	42.3	0.422	0.098	90.17 $\pm$ 0.14

Table 2. MST depth, number of parameters, bit-Ops, and accuracy *w.r.t.* different values of λ on CIFAR-10 VGG-small model.

Hardware implementation: The inference hardware architecture is implemented on a Xilinx FPGA platform called xczu3eg-sbva484 part. Before programming to FPGA, the design is run with the corresponding C/C++ model and simulated via Synopsys VCS. Finally, Vivado 2018.3 is used for the design synthesis and implementation.

5.2. Ablation Studies

In this section, experimental results and discussion related to hyper-parameter λ, and γ are described.

Effect of hyper-parameter γ: As explained in Sec. 4.1, an additional loss related to MST distance is added for the learning. The effect of the loss depends on the γ, while the γ changes following the epoch. To facilitate the evaluation, Figure 7 provides MST distance curves for all binary convolution layers and training-validation accuracy curves in the entire learning process. In the first half-time, the γ is still significant, and the training focuses on MST distance optimization, while the accuracy optimization is temporarily less affected. Consequently, the MST distance of all binary convolutions is significantly down, especially for the deeper layers with more channels, while the training accuracy suddenly decreases and the validation accuracy goes down and strongly fluctuates at this time. In the second half-time, the training focuses on accuracy optimization with the lower γ value. Thus, the training accuracy tends to increase, while the validation accuracy rises, too, with a lower fluctuation. Although the MST distance also escalates again on all layers, these values are lower than at the initial time, especially on deeper layers. This is the reason why we can save computational costs by using this approach.

Effect of hyper-parameter λ: The hyper-parameter λ helps scale down the effect of the total MST distance to the new loss function. Each model has a different number of layers and channels that directly affects the total MST distance. Therefore, depending on the model, we adjust this hyper-parameter to balance the loss for accuracy and MST distance. In this section, the VGG-small model and CIFAR-10 dataset are used to estimate how the λ impacts the number of parameters, bit-operations, MST-depth, and accuracy with the range of λ from 1e-7 to 1e-5. Table 2 and Figure 8 provide the number of parameters, bit-operations, MST-depth, accuracy, and the number parameters for each binary convolution layer *w.r.t* different λ values. Accordingly, the number of parameters, bit-operations, and accuracy tend to be lower with higher λ values. However, the number of parameters/bit-ops is considerably reduced at 3.3×, while the accuracy degradation is acceptable with 2% when increasing the λ from 1e-7 to 1e-5. Besides, the MST depth also decreases when the λ value increases. Especially, when increasing λ from 1e-7 to 5e-7, the depth reduce 1.67×.

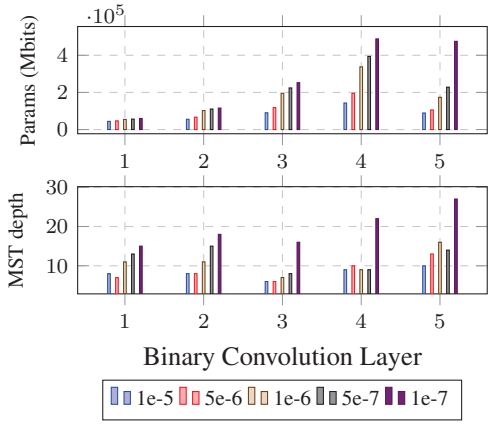


Figure 8. Number of parameters and MST depth on each convolution layer *w.r.t.* different λ values.

5.3. Comparison with SOTA methods

To demonstrate the effectiveness of the proposed method, we perform a series of training experiments with various NN models on CIFAR-10, and ImageNet, then compare them with state-of-the-art methods. Specifically, for each NN model, we provide two experimental results corresponding to two cases 1) apply both the learning optimization in Sec. 4.2 and the MST reordering to the final training result (**Ours 1**); 2) directly apply the MST reordering to pre-trained BNN models (**Ours 2**). In this paper, the pre-trained BNN models we use for fine-tuning and exploring the MST are provided by Xu *et al*. [33]. In terms of acceleration, a neural network model is fully trained on CIFAR-10 and implemented on an FPGA hardware platform to compare with recent hardware architectures.

Software training comparison: For CIFAR-10, we

N-work	Method	#Params (Mbit)	#Bit-Ops (GOps)	Top-1 Acc. (%)
VGG small	RAD[8]	4.571	0.603	90.0
	IR-Net[24]	4.571	0.603	90.4
	RBNN[20]	4.571	0.603	91.3
	Adabin[28]	4.571	0.603	92.3
	ReCU[33]	4.571	0.603	92.4
	SNN[32]	3.047	0.194	91.0
	Ours 1|2	0.556|1.611	0.122|0.232	91.5|93.3*
ResNet 18	RAD[8]	10.99	0.547	90.5
	IR-Net[24]	10.99	0.547	91.5
	RBNN[20]	10.99	0.547	92.2
	ReCU[33]	10.99	0.547	92.8
	Adabin[28]	10.99	0.547	93.1
	SNN[33]	7.324	0.289	91.0
	Ours 1|2	0.814|4.293	0.104|0.216	91.6|93.2*
ResNet 20	DSQ[11]	0.267	0.040	84.1
	IR-Net[24]	0.267	0.040	86.5
	RBNN[20]	0.267	0.040	86.5
	ReCU[33]	0.267	0.040	87.4
	Adabin[28]	0.267	0.040	88.2
	SNN[32]	0.178	0.040	85.1
	Ours 1|2	0.096|0.116	0.015|0.017	86.5|88.0*

* Accuracy after fine-tuning is at https://github.com/z-hXu/ReCU [33].

Table 3. Comparison with the state-of-the-art methods on CIFAR-10. The bit-width is 1 for both activation and weight.

conduct experiments on three neural BNN models: RestNet-18/20 and VGG-small. we compare our results with RAD [8], IR-Net [24], Adabin [28], ReCU [33], DSQ [11], and SNN [32]. Specifically, as shown in Table 3, when comparing **Ours 1** with the highest-accuracy methods on all models, our proposed method reduces $8.2\times$ parameters and $4.9\times$ bit-Ops with a 0.97% accuracy drop on VGG-small. Meanwhile, when directly exploring the MST on the pre-trained models, **Ours 2** gets $2.8\times$ parameters and $2.6\times$ bit-Ops reduction without accuracy degradation. On ResNet-18, **Ours 1** gives $13.50\times$ parameters, $5.26\times$ bit-Ops reduction with a 1.29% accuracy drop, while **Ours 2** reduces $2.6\times$ parameters and $2.5\times$ bit-Ops without compromising accuracy. Additionally, we have $2.78\times$ parameters, $2.67\times$ bit-Ops reduction, and a 1% accuracy drop with **Ours 1**; $2.3\times$ parameters, $2.35\times$ bit-Ops reduction and 0% accuracy drop on ResNet-20.

Compared **Ours 1** with the compression method SNN [32] on all models, the proposed method obtains a better compression ratio with a maximum reduction of $9.0\times$ parameters, $2.78\times$ bit-Ops on ResNet-18 while yielding higher accuracy and up to maximum 1.4% on ResNet-20. Meanwhile, with **Our 2**, we reduce maximum $1.89\times$ parameters on VGG-small, $2.35\times$ bit-Ops on ResNet-20, while achieving higher accuracy up to maximum 2.9% on ResNet-20.

For ImageNet, experiments are implemented on ResNet-18/34 and comparison is performed with BNN+ [16], Bi-Real [23], XNOR++ [4], IR-Net [24], Adabin [28], ReCU [33], SNN [32]. According to Table 4, **Ours 1** reduces $3.2\times$ parameters, $2.6\times$ bit-Ops on ResNet-18; and $2.23\times$ parameters, $2.27\times$ bit-Ops on ResNet-34, compared to the baseline [33], while the accuracy degradation is 4% and 2.2% on ResNet18 and ResNet-34, respectively. **Ours 2** achieves $2.27\times$ parameters, $2.34\times$ bit-Ops reduction on ResNet-18; and $2.2\times$ parameters, $2.26\times$ bit-Ops reduction without accuracy drop. Compared to SNN with the highest compression ratio recently, our proposed method with **Ours 1** saves $2.13\times$ parameters and $1.3\times$ bit-Ops on ResNet-18, $1.49\times$ parameters and $1.09\times$ bit-Ops on ResNet-34, while the trained models give higher accuracy of 0.7% and 1.5% on ResNet-18, ResNet-34, respectively. Meanwhile, **Ours 2** without accuracy drop saves up to $1.51/1.47\times$ parameters and $1.23/1.08\times$ bit-Ops on ResNet-18/34, respectively.

N-work	Method	#Params (Mbit)	#Bit-Ops (GOps)	Top-1 Acc. (%)
ResNet 18	BNN+[16]	10.99	1.677	53.0
	Bi-Real[23]	10.99	1.677	56.4
	XNOR++[4]	10.99	1.677	57.1
	IR-Net[24]	10.99	1.677	58.1
	Adabin[28]	10.99	1.677	63.1
	ReCU[33]	10.99	1.677	61.0
	SNN[32]	7.32	0.883	56.3
	Ours 1|2	3.43|4.84	0.636|0.716	57.0|61.2*
ResNet 34	Bi-Real[23]	21.09	3.526	62.2
	IR-Net[24]	21.09	3.526	62.9
	Adabin[28]	21.09	3.526	66.4
	ReCU[33]	21.09	3.526	65.1
	SNN[32]	14.06	1.696	61.4
	Ours 1|2	9.44|9.51	1.550|1.558	62.9|65.4*

* Accuracy after fine-tuning is at https://github.com/z-hXu/ReCU [33].

Table 4. Comparison with the state-of-the-art methods on ImageNet. The bit-width is 1 for both activation and weight.

Hardware implementation comparison: Table 5 provides the hardware performance comparison with previous works using CIFAR-10 for the training. we compare the proposed architecture with FINN [30], FINN-R [3] from Xilinx, ReBNet [10] and the design in [31]. Accordingly, the proposed accelerator takes the leading place in both speed and area efficiency (FPS/LUTs). More specifically, it performs $1.6\times$ faster than the fastest previous work (FINN), while obtaining $3.7\times$ area efficiency with a little higher accuracy. In addition, compared to [31], our method gains over $1.81\times$ area efficiency. Besides, to compare our method with the K-mean approach, we implement another design from the same training result with the K-mean method. According to Table 5, our method performs better $1.25\times$ in terms of resources and area efficiency.

6. Conclusion

In conclusion, this paper introduces a comprehensive method for compressing BNNs, which covers the entire process from learning to hardware implementation. By utilizing the minimum spanning tree (MST), we effectively re-

Design	Freq (MHz)	LUTs	Acc. (%)	FPS (K)	FPS/ LUTs
FINN[30]	200	46,253	80.1	22	0.47
FINN[30]	125	365,963	80.1	128	0.35
FINN-R[3]	237	332,637	80.1	105	0.31
FINN-R[3]	300	41,733	80.1	20	0.48
FINN[3]	300	25,431	80.1	1.9	0.07
ReBNet[10]	200	53,200	80.6	6	0.11
[31]	210	290,012	80.2	205	0.70
Ours (K-mean)	210	201,434	80.5	205	1.01
Ours (MST)	**210**	**161,294**	**80.5**	**205**	**1.27**

Table 5. Hardware performance comparison with the state-of-the-art architectures on CIFAR-10.

duce the computation cost of binary convolution calculation. To optimize the MST during the training stage, a learning algorithm is proposed. Moreover, we present a hardware implementation for the inference task. Our experimental results demonstrate that the proposed method significantly reduces computation costs while maintaining high accuracy, making it promising for practical applications.

7. Acknowledgements

This work was supported by the NRF grant funded by the Korea government(MSIT) (No. RS-2023-00207816), and part by IITP Grant funded by MSIT (Artificial Intelligence Innovation Hub) under Grant 2021-0-02068, (No. RS-2022-00155911, AI Convergence Innovation Human Resources Development (Kyung Hee University)), IITP No.2019-0-01287, Evolvable Deep Learning Model Generation Platform for Edge Computing and ITRC support program (IITP-2023-RS-2023-00258649).

References

[1] Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM (JACM)*, 9(1):61–63, 1962. 2, 5

[2] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. 3

[3] Michaela Blott, Thomas B Preußer, Nicholas J Fraser, Giulio Gambardella, Kenneth O'brien, Yaman Umuroglu, Miriam Leeser, and Kees Vissers. 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):1–23, 2018. 2, 8, 9

[4] Adrian Bulat and Georgios Tzimiropoulos. Xnor-net++: Improved binary neural networks. *arXiv preprint arXiv:1909.13863*, 2019. 1, 3, 8

[5] Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. Deep learning with low precision by half-wave gaussian quantization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5918–5926, 2017. 2

[6] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits*, 52(1):127–138, 2016. 1

[7] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016. 1, 2

[8] Ruizhou Ding, Ting-Wu Chin, Zeye Liu, and Diana Marculescu. Regularizing activation distribution for training binarized deep networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11408–11417, 2019. 8

[9] Cheng Fu, Shilin Zhu, Hao Su, Ching-En Lee, and Jishen Zhao. Towards fast and energy-efficient binarized neural network inference on fpga. *arXiv preprint arXiv:1810.02068*, 2018. 1, 2, 3, 4

[10] Mohammad Ghasemzadeh, Mohammad Samragh, and Farinaz Koushanfar. Rebnet: Residual binarized neural network. In *2018 IEEE 26th annual international symposium on field-programmable custom computing machines (FCCM)*, pages 57–64. IEEE, 2018. 2, 8, 9

[11] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4852–4861, 2019. 8

[12] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 1

[13] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)*, 28(1):100–108, 1979. 2, 4

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1, 2, 5

[15] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015. 1

[16] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. *Advances in neural information processing systems*, 29, 2016. 8

[17] Hyeonuk Kim, Jaehyeong Sim, Yeongjae Choi, and Lee-Sup Kim. A kernel decomposition architecture for binary-weight convolutional neural networks. In *Proceedings of the 54th Annual Design Automation Conference 2017*, pages 1–6, 2017. 1, 3

[18] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 2

[19] Dongsoo Lee, Se Jung Kwon, Byeongwook Kim, Yongkweon Jeon, Baeseong Park, and Jeongin Yun. Flexor: Train-

able fractional quantization. *Advances in neural information processing systems*, 33:1311–1321, 2020. 3

[20] Mingbao Lin, Rongrong Ji, Zihan Xu, Baochang Zhang, Yan Wang, Yongjian Wu, Feiyue Huang, and Chia-Wen Lin. Rotated binary neural network. *Advances in neural information processing systems*, 33:7474–7485, 2020. 1, 3, 8

[21] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. *Advances in neural information processing systems*, 30, 2017. 3

[22] Zechun Liu, Zhiqiang Shen, Marios Savvides, and Kwang-Ting Cheng. Reactnet: Towards precise binary neural network with generalized activation functions. In *European conference on computer vision*, pages 143–159. Springer, 2020. 1, 3

[23] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In *Proceedings of the European conference on computer vision (ECCV)*, pages 722–737, 2018. 1, 3, 8

[24] Haotong Qin, Ruihao Gong, Xianglong Liu, Mingzhu Shen, Ziran Wei, Fengwei Yu, and Jingkuan Song. Forward and backward information retention for accurate binary neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2250–2259, 2020. 3, 8

[25] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016. 1, 3

[26] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015. 2

[27] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. 1

[28] Zhijun Tu, Xinghao Chen, Pengju Ren, and Yunhe Wang. Adabin: Improving binary neural networks with adaptive binary sets. In *European Conference on Computer Vision*, pages 379–395. Springer, 2022. 3, 8

[29] William Thomas Tutte and William Thomas Tutte. *Graph theory*, volume 21. Cambridge university press, 2001. 4

[30] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays*, pages 65–74, 2017. 1, 2, 8, 9

[31] Quang Hieu Vo, Ngoc Linh Le, Faaiz Asim, Lok-Won Kim, and Choong Seon Hong. A deep learning accelerator based on a streaming architecture for binary neural networks. *IEEE Access*, 10:21141–21159, 2022. 1, 2, 3, 4, 5, 6, 8, 9

[32] Yikai Wang, Yi Yang, Fuchun Sun, and Anbang Yao. Sub-bit neural networks: Learning to compress and accelerate binary neural networks. In *International Conference on Computer Vision (ICCV)*, 2021. 1, 2, 3, 8

[33] Zihan Xu, Mingbao Lin, Jianzhuang Liu, Jie Chen, Ling Shao, Yue Gao, Yonghong Tian, and Rongrong Ji. Recu: Reviving the dead weights in binary neural networks. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5198–5208, 2021. 1, 2, 3, 5, 6, 7, 8