A Convolutional Result Sharing Approach for Binarized Neural Network Inference

Ya-Chun Chang, Chia-Chun Lin, Yi-Ting Lin, Yung-Chih Chen[§], and Chun-Yao Wang
Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, R.O.C.
[§]Department of Computer Science and Engineering, Yuan Ze University, Chungli, Taiwan, R.O.C.

***Abstract*—The binary-weight-binary-input binarized neural network (BNN) allows a much more efficient way to implement convolutional neural networks (CNNs) on mobile platforms. During inference, the multiply-accumulate operations in BNNs can be reduced to XNOR-popcount operations. Thus, the XNOR-popcount operations dominate most of the computation in BNNs. To reduce the number of required operations in convolution layers of BNNs, we decompose 3-D filters into 2-D filters and exploit the repeated filters, inverse filters, and similar filters to share results. By sharing the results, the number of operations in convolution layers of BNNs can be reduced effectively. Experimental results show that the number of operations can be reduced by about 60% for CIFAR-10 on BNNs while keeping the accuracy loss within 1% of originally trained network.**

***Index Terms*—convolutional neural network, binarized neural network, approximate computing**

I. Introduction

Deep convolutional neural networks (DCNNs) have been widely used in the artificial intelligence (AI) field. In many applications such as computer vision, speech recognition, and robotics, DCNNs achieve superior accuracy. Accordingly, there are great demands for using DCNNs on mobile platforms such as smartphones, wearable devices, and IOT devices to meet people's needs in life. However, DCNNs contain vast amounts of parameters and operations needed to perform, making it difficult to implement DCNNs on mobile platforms. This is because mobile platforms generally offer limited storage space and are battery constrained.

To solve these pressing issues, many studies have been proposed to improve energy efficiency and reduce the model size of DCNNs. Related works include reducing precision [5] [6] [7], network pruning [8] [9], and weight sharing [10] [11]. As for reducing precision, the precision can be extremely reduced to one bit. This topic is usually referred to as binarized neural networks (BNNs) [1] [2] [4]. A BNN is essentially a convolutional neural network (CNN) whose weights or both weights and inputs are binarized to single bits. BinaryConnect (BC) [1] proposed the concept of binary weights. With binary weights, there is no longer a need for multiplication in the operations of convolution during inference. Besides, the network size becomes significantly smaller than an equivalent network with full precision since each weight can be represented by only one bit. Afterward, BinaryNet [2] and XNOR-Net [4] binarize weights as well as inputs. Binary weights and inputs allow a much more efficient way to implement convolutions on mobile platforms because the original operations, which include multiplications, additions, and subtractions, can be further reduced to XNOR and popcount operations. This paradigm shift empowers neural networks with deep convolution layers to perform real-time inference on mobile devices.

Focusing on BNNs, there were studies exploring properties of BNNs to improve their inference performances. In [12], redundant operations were skipped in binary-weight BNNs by a kernel decomposition architecture, thereby reducing the energy consumption. [13] proposed an algorithm to achieve logic sharing of dense layers in a BNN for reducing look-up table usage and net counts on FPGA implementation. Besides, [14] proposed three BNN-specific optimizations to enable a hardware-friendly implementation of binary-weight-binary-input BNNs. However, the XNOR-popcount operations still dominate most of the computation in convolution layers of a BNN. Popcount operations are time and energy-consuming since they sum up bits continuously to generate the results.

Therefore, we propose a two-dimensional (2-D) filter's convolutional result sharing approach for convolution layers of binary-weight-binary-input BNNs. We decompose 3-D filters into a set of 2-D filters and exploit *repeated filters*, *inverse filters*, and *similar filters* to share the convolutional results. Thereafter, the number of required XNOR-popcount operations during inference can be reduced effectively. Among these three kinds of filters, repeated filters and inverse filters are regarded as filter repetitions. With filter repetitions, we can exactly share the convolutional results without affecting the network accuracy. Similar filters are used for filter approximation, which would lead to small accuracy loss.

II. Backgrounds

A. Convolutional Neural Networks

A common form of deep neural networks is composed of multiple convolution layers, usually called deep convolutional neural networks (DCNNs). Fig. 1 shows how a convolution layer works. In computation, it takes the $C \times H \times W$ feature map matrix as its input feature maps (ifmaps). Ifmaps are convolved with each filter of size $C \times K \times K$ to generate each channel of the output feature maps (ofmaps) by running

This work is supported in part by the Ministry of Science and Technology of Taiwan under MOST 106-2221-E-007-111-MY3, MOST-106-2314-B-007-005, MOST 106-2622-8-007-015-TA, MOST 107-2622-8-007-014-TA, and MOST 108-2221-E-155-047.

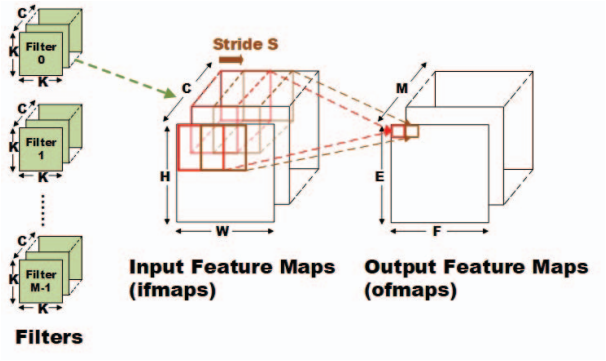


Fig. 1: Illustration of a convolution layer.

Algorithm 1: The Naive Convolution Algorithm

```
for m = 0 to M − 1 do
    for e = 0 to E − 1 do
        for f = 0 to F − 1 do
            for c = 0 to C − 1 do
                ofmaps[m][e][f] +=
                  Σ_{i=0}^{K−1} Σ_{j=0}^{K−1} filters [m][c][i][j] × ifmaps [c][e × S
                  + i][f × S + j]
            end
        end
    end
end
```

Fig. 2: Pseudocode for a convolution layer.

through all the sliding windows with a stride S, and each channel of ofmaps has the size of $E \times F$. Thus, M filters produce M-channel ofmaps. The pseudocode of the computation of a convolution layer is expressed in Fig. 2.

B. Binarized Neural Networks

The authors in [2] proposed a method to train BNNs with binary weights and binary inputs. Both weights and inputs are constrained to either +1 or -1. The inference can be performed on XNOR-popcount operations when the domain of weight {+1, -1} is mapping to Boolean domain {1, 0}. With the binary weights, memory size and the number of memory accesses can be significantly decreased in BNNs. Besides, because BNNs can replace multiply-accumulate operations with XNOR-popcount operations, the computation effort can be decreased dramatically.

In addition to these benefits of BNNs, [3] mentioned that BNNs can lead to filter repetitions. With the binary weights, the number of unique filters is bounded by the filter size. For example, for a 3×3 2-D filter, the maximal number of unique 2-D filters is 2^9 = 512. Additionally, an inverse filter can be treated as a repetition as well because its result is equal to that of the original filter multiplied by -1. For example, the filter [-1,1,1] is the inverse of the filter [1,-1,-1]. When considering the inverse filters, the maximal number of unique 2-D filters in the last example is reduced to $2^9/2 = 256$.

III. Proposed Approach

A. Convolutional Result Sharing with Filter Repetitions

We exploit filter repetitions to share convolutional results, thereby reducing the number of operations in BNNs. In our work, two kinds of filters, repeated filters and inverse filters, are regarded as the filter repetitions. The repeated filters are filters with exactly the same weights while the inverse filters are filters with completely opposite weights.

1) 2-D Filter ID: As mentioned in Section II-B, for a $K \times K$ 2-D filter, the maximal number of unique 2-D filters is $2^{K^2}/2 = 2^{(K^2-1)}$ in BNNs. The filters' weights in BNNs, which are originally in the domain {+1, -1}, are encoded to Boolean domain {1, 0} in this work. For a better explanation, every unique 2-D filter is denoted as FID_p with respect to its weights. For example, given a 3×3 unique 2-D filter with the row patterns [0,0,0], [0,0,0], [1,1,1], we concatenate these three rows as a 3^2-bit binary number 000000111, which is 7 in decimal. Thus, this unique 2-D filter is denoted as FID_7.

2) Inverse Bit: Since we consider the inverse filters, an FID corresponds to two filters whose weights are completely opposite to each other. We use an inverse bit (IVB) to distinguish them. For example, the filter [[1,1,1], [1,1,1], [0,0,0]] is the inverse of FID_7 while the filter [[0,0,0], [0,0,0], [1,1,1]] is the original of FID_7. Thus, the IVB of the former filter is 1 and the IVB of the latter is 0. Given a $K \times K$ filter, the convolutional result, CO and CI, of an original filter and an inverse filter are expressed in EQs (1) and (2), respectively,

$$CO = \sum_{n=0}^{K^2-1}(I_n \odot W_n) \quad (1)$$

$$CI = \sum_{n=0}^{K^2-1}(I_n \odot \overline{W_n}) \quad (2)$$

where I_n is the n^{th} bit of input, W_n is the n^{th} bit of an original filter's weights, $\overline{W_n}$ is the n^{th} bit of an inverse filter's weights, and $\odot$ is a bit-wise XNOR operation. We observed that the convolutional result of a $K \times K$ inverse filter can be deduced from the convolutional result of its original filter.

3) Decomposition from 3-D to 2-D: The actual filters are usually 3-D. The maximal number of unique 3-D filters is enormous because it exponentially grows with the number of bits in a 3-D filter. To handle this problem, 3-D filters are decomposed into sets of 2-D filters in our work. In Fig. 3, there are filters, ifmaps, and ofmaps, which have been previously presented in Fig. 1. After decomposing 3-D filters into 2-D filters, there are $(M \cdot C)$ 2-D filters. Every 2-D filter is indexed by FT_{mc}, where m represents the m^{th} filter, and c represents the c^{th} channel of the filter. The $C \times K \times K$ matrix at a sliding position of ifmaps is referred to as the 3-D input matrix. The $M \times 1 \times 1$ matrix at a sliding position of ofmaps is referred to as the 3-D output matrix. To match the input size of 2-D filters, the 3-D input matrix is decomposed into C 2-D input matrices. Every 2-D input matrix is indicated by IN_c, where c represents the c^{th} channel of ifmaps. The 3-D output matrix is also decomposed into M 2-D output matrices. Every 2-D output matrix is indicated by OUT_m, where m represents the m^{th} channel of ofmaps.

To generate an OUT_m, C 2-D filter's convolutional results are accumulated. Each 2-D filter's convolutional result is obtained by conducting an XNOR-popcount operation on IN_c with FT_{mc}. We use EQ (3) to represent this idea,

$$OUT_m = \sum_{c=0}^{C-1} IN_c * FT_{mc} \quad (3)$$

where $*$ is an XNOR-popcount operation for a 2-D filter.

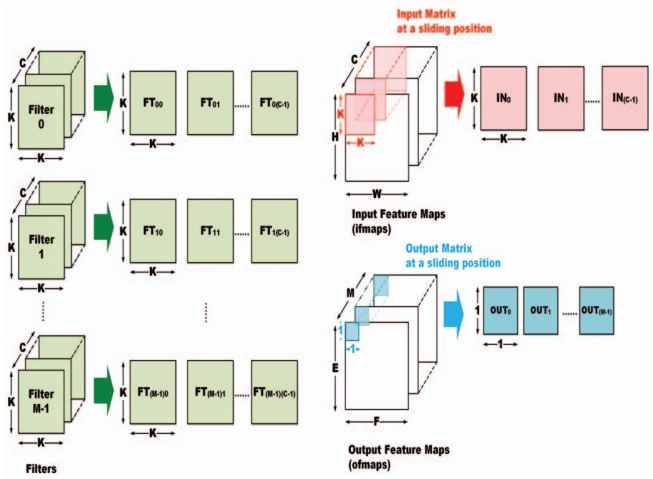
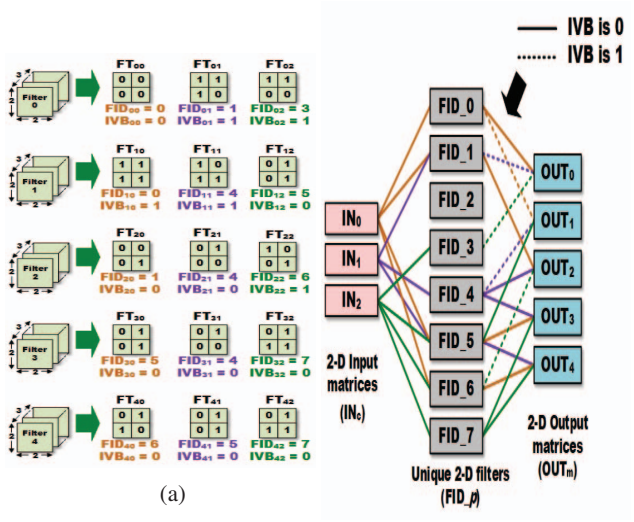


Fig. 3: Matrix decomposition from 3-D to 2-D.



(a)

(b)

Fig. 4: Example of filter dependency graph. (a) Five $3 \times 2 \times 2$ filters are decomposed to 15 2-D filters. (b) Its corresponding filter dependency graph.

4) Filter Dependency Graph: Every FT_{mc} has an FID and an IVB with respect to its filter weights, which are denoted as FID_{mc} and IVB_{mc}. For instance, if a 3×3 filter FT_{00} is with [[1,1,1], [1,1,1], [0,0,0]], the FID_{00} is 7, and the IVB_{00} is 1. These FIDs and IVBs can be attained by exploring the filter weights from a trained network. For filter repetitions, filters with the same FIDs and the same IVBs are repeated filters. Filters with the same FIDs but different IVBs are inverse filters. We exploit these two kinds of filters in our sharing approach. Thus, as long as filters have the same FID, the convolutional results of those filters will be shared. Consequently, after having FIDs and IVBs of all FT_{mc}, EQ (3) becomes EQ (4),

$$OUT_m = \sum_{c=0}^{C-1} POP_{mc} \tag{4}$$

where POP_{mc} is given by EQ (5).

$$\text{POP}_{mc} = \begin{cases} IN_c * \text{FID\_}FID_{mc} & \text{, if } IVB_{mc} \text{ is 0} \\ K^2 - (IN_c * \text{FID\_}FID_{mc}) & \text{, if } IVB_{mc} \text{ is 1} \end{cases} \tag{5}$$

Fig. 4(a) illustrates an example of five $3 \times 2 \times 2$ filters in a convolution layer. FIDs and IVBs of these 15 2-D filters are also shown. With these 2-D filters, we construct the filter dependency graph to display the relation among the filters, 2-D input matrices, and 2-D output matrices. Moreover, the filter dependency graph explains how filter repetitions can reduce the amount of operations. The filter dependency graph in Fig. 4(b) corresponds to the example in Fig. 4(a). Since the 2-D filter size is 2×2, the maximal number of unique 2-D filters is $2^{(2^2-1)} = 8$. FID_0, FID_1, ..., and FID_7 in Fig. 4(b) represent the 8 unique 2-D filters. IN_0, IN_1, and IN_2 in Fig. 4(b) represent 2-D input matrices. OUT_0, OUT_1, ..., and OUT_4 in Fig. 4(b) represent 2-D output matrices. A line linking an input matrix IN_c and a unique 2-D filter FID_p means that IN_c has to convolve with FID_p. A line linking a unique 2-D filters FID_p and an output matrix OUT_m means that the convolutional result of FID_p with one of the 2-D input matrices IN_c is needed by OUT_m. If IVB_{mc} is 0, a solid line is used to connect them. Otherwise, a dotted line is used to connect them. Different line colors represent that they are sourced from different 2-D input matrices. According to EQ (3), $OUT_0 = IN_0 * FT_{00} + IN_1 * FT_{01} + IN_2 * FT_{02}$. Thus, there exist lines from IN_0 to FID_FID_{00}(=FID_0), IN_1 to FID_FID_{01}(=FID_1), and IN_2 to FID_FID_{02}(=FID_3). These lines are then connected to OUT_0 from FID_0, FID_1, and FID_3. From this graph, we can see that OUT_0 and OUT_1 have a brown line connecting to FID_0, which means both OUT_0 and OUT_1 need IN_0 to be convolved with FID_0. This is because the filter repetition occurs on FT_{00} and FT_{10}. In this case, $IN_0 *$ FID_0 can be computed only once and shared to those 2-D output matrices requesting the result.

5) Convolutional Result Sharing: We define two sets, SO_{cp} and SFT_{cp}, to describe how the 2-D filter's convolutional results are shared explicitly. SO_{cp} is the set of all OUT_m such that FID_{mc} is p, and is formally defined in EQ (6),

$$SO_{cp} = \{OUT_m \mid FID_{mc} = p, 0 \leq m \leq M-1, m \in \mathbb{Z}\} \tag{6}$$

where M is the number of 2-D output matrices. The element OUT_m in the set SO_{cp} represents that OUT_m needs IN_c to be convolved with FID_p. For example, SO_{00} = $\{OUT_0, OUT_1\}$, SO_{01} = $\{OUT_2\}$, and SO_{02} = $\emptyset$ in Fig. 4(b). The number of elements in the set SO_{cp}, denoted by $|SO_{cp}|$, is the number of 2-D output matrices that need the term $IN_c *$ FID_p for calculation. SFT_{cp} is the set of all FT_{mc} such that FID_{mc} is p, and is formally expressed in EQ (7),

$$SFT_{cp} = \{FT_{mc} \mid FID_{mc} = p, 0 \leq m \leq M-1, m \in \mathbb{Z}\} \tag{7}$$

where M is the number of 2-D output matrices. The element FT_{mc} in the set SFT_{cp} represents that its FID is p. For example, SFT_{00} = $\{FT_{00}, FT_{10}\}$, SFT_{01} = $\{FT_{20}\}$, and SFT_{02} = $\emptyset$ in Fig. 4(a). If the number of elements in the set SFT_{cp}, denoted by $|SFT_{cp}|$, is greater than one, the filter

repetitions occur on all filters in the set SFT_{cp}. This is because when taking IN_c as an input, all filters FT_{mc} in the set SFT_{cp} have the same FID of p. In the convolutional result sharing approach, IN_c* FID_p is computed only once and shared to all OUT_m in SO_{cp}. In this way, the number of XNOR-popcount operations can be reduced. This sharing approach does not change the functionality of a trained network.

6) Reduction on XNOR-Popcount Operations: To estimate the reduction on XNOR-popcount operations, we define another kind of set SID_c. SID_c is the set of FID p such that FID_{mc} is p, and is formally represented in EQ (8),

$$SID_c = \{p \mid p = FID_{mc}, 0 \leq m \leq M-1, m \in \mathbb{Z}\} \quad (8)$$

where M is the number of 2-D output matrices. The element p in the set SID_c represents that IN_c has to convolve with the unique 2-D filter FID_p. The number of elements in the set SID_c, denoted by $|SID_c|$, is the number of unique 2-D filters with which IN_c needs to convolve.

Without the convolutional result sharing, there are $(M \cdot C)$ 2-D filters that need to convolve with the 2-D input matrices at each sliding position. However, if the convolutional result sharing approach is applied, there are $\sum_{c=0}^{C-1} |SID_c|$ filters that need to convolve with the input. The more filter repetitions occur, the smaller $\sum_{c=0}^{C-1} |SID_c|$ is. The reduction ratio of XNOR-popcount operations is then defined as EQ (9).

$$Reduction\ ratio = 1 - \frac{\sum_{c=0}^{C-1} |SID_c|}{M \cdot C} \quad (9)$$

B. Convolutional Result Sharing with Filter Approximation

In Section III-A, we use repeated filters and inverse filters to share convolutional results, which means that the filters with exactly the same weights or completely opposite weights are exploited. However, in this subsection, we further exploit the filters with different weights to create more opportunities for convolutional result sharing.

We observed that given a 2-D input matrix, there may be more than one unique 2-D filter that can produce the same convolutional result. For example, with a 2×2 2-D input matrix IN_0 = [[0, 0], [0, 0]], there are three unique 2-D filters that can produce the same convolutional result of 3 such as FID_1 = [[0, 0], [0, 1]], FID_2 = [[0, 0], [1, 0]], and FID_4 = [[0, 1], [0, 0]]. In this example, if IN_0 has a higher probability of being [[0, 0], [0, 0]], filters with FID of 1, 2, and 4 can be seen as similar filters. We will apply this concept of approximation to similar filters for sharing more convolutional results.

1) Degree of Similarity of Filters: We define the Degree of Similarity (DSIM) of 2-D filters to evaluate how much similarly any two unique filters FID_p_i and FID_p_j behave for each channel c of ifmaps. The DSIM between FID_p_i and FID_p_j is denoted as $DSIM_{cp_ip_j}$. $DSIM_{cp_ip_j}$ represents the percentage that $IN_c * FID\_p_i$ and $IN_c * FID\_p_j$ are equal, and is expressed as EQ (10),

$$DSIM_{cp_ip_j} = \frac{EQCount_{cp_ip_j}}{E \times F \times N} \quad (10)$$

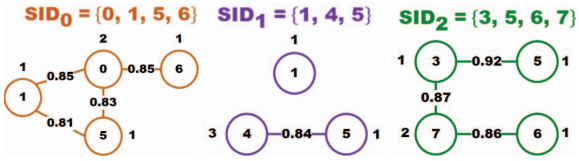
where $EQCount_{cp_ip_j}$ is the number of occurrences that IN_c* FID_p_i and $IN_c * FID\_p_j$ are equal. Since the number of equality depends on the input data of a BNN, this number is obtained by performing forward propagation in a BNN with N samples of training data. With N samples, there are $E \times F \times N$ sliding positions totally on a channel of ifmaps.

Besides, a threshold value about the degree of similarity (DSIMTH) is set to determine whether filters with different FIDs can be regarded as similar filters. If $DSIM_{cp_ip_j}$ reaches the DSIMTH, filters in SFT_{cp_i} and SFT_{cp_j} are considered as similar filters. The DSIMTH can be adjusted by users depending on networks' redundancy or the accuracy loss users can tolerant. A lower value of DSIMTH means that more filters can be considered as similar filters. Thus, the accuracy loss may be rising. Contrarily, a higher value of DSIMTH would not lead to much accuracy degradation. For two FIDs p_i and p_j, and a channel c of ifmaps, if filters in SFT_{cp_i} and filters in SFT_{cp_j} are identified as similar filters, either filters in SFT_{cp_i} can be approximated to have FIDs of p_j or filters in SFT_{cp_j} can be approximated to have FIDs of p_i. If the filters in SFT_{cp_i} are approximated to have FIDs of p_j, the term IN_c* FID_p_i will not be conducted anymore and the 2-D output matrices in SO_{cp_i} will receive the result of IN_c* FID_p_j. This result that 2-D output matrices in SO_{cp_i} receive is called the approximate 2-D convolutional result. On the other hand, the result that 2-D output matrices in SO_{cp_j} receive is called the exact 2-D convolutional result.

2) Degree of Importance of Filters: Considering the accuracy degradation after approximation, we maximize the percentage that 2-D output matrices receive exact 2-D convolutional results instead of approximate ones. For each channel c of ifmaps, every FID_p has a Degree of Importance (DI), denoted as DI_{cp}. DI_{cp} is defined as the number of 2-D output matrices that need the convolutional result of FID_p with the 2-D input matrix IN_c. Thus, it can be obtained by EQ (11). A higher DI_{cp} means that filters in SFT_{cp} have a higher priority approximating other filters instead of being approximated.

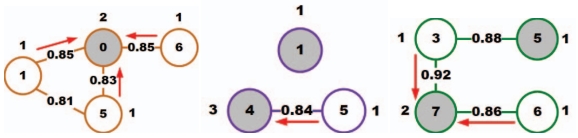
$$DI_{cp} = |SO_{cp}| = |SFT_{cp}| \quad (11)$$

3) Filter Similarity Graph: We construct a filter similarity graph to solve the filter approximation. The graph for the c^{th} input matrix contains all vertices p in SID_c, and the weight of each vertex p is DI_{cp}. For every $DSIM_{cp_ip_j}$ greater than or equal to DSIMTH, an undirected edge $\{p_i, p_j\}$ with the weight $DSIM_{cp_ip_j}$ is added. In a filter similarity graph of the c^{th} 2-D input matrix, two adjacent vertices p_i and p_j imply that filters in SFT_{cp_i} and SFT_{cp_j} are identified as similar filters. From the perspective of the vertices, every vertex can approximate its adjacent vertices or be approximated by one of its adjacent vertices. Therefore, for every vertex, at least one of its adjacent vertices or itself should be chosen. If a vertex p_i is chosen, the filters in SFT_{cp_i} are not approximated. In contrast, if a vertex p_i is not chosen, the filters in SFT_{cp_i} are approximated, and one of its chosen adjacent vertices has to approximate the vertex p_i. We take the filters in Fig. 4(a) as our example again. The filter similarity graphs of the IN_0, IN_1, and IN_2 are shown in Fig. 5, where all the edge weights are the assumed values for the explanation.



(a) (b) (c)

Fig. 5: Filter similarity graphs corresponding to the example in Fig. 4(a). (a) For IN_0. (b) For IN_1. (c) For IN_2.



(a) (b) (c)

Fig. 6: The result from the filter similarity graphs of Fig. 5. (a) For IN_0. (b) For IN_1. (c) For IN_2.

4) Filter Approximation: We solve the filter approximation problem by using the filter similarity graph. We have a constraint and three objectives.

Constraint : For every vertex, at least one of its adjacent vertices or itself should be chosen.

Objective 1 : The number of unique 2-D filters with which every 2-D input matrix needs to convolve is minimized. That is, $\sum_{c=0}^{C-1} |SID_c|$ is minimized, or the total number of chosen vertices in the filter similarity graphs should be minimized.

Objective 2 : Under the premise of Objective 1, the percentage that 2-D output matrices receive exact 2-D convolutional results is maximized. That is, the total vertex weights of chosen vertices are maximized.

Objective 3 : Based on Objective 2, for the approximated filter, it is approximated by the most similar filter. That is, for each unchosen vertex, one of its chosen adjacent vertices with the greatest edge weight approximates the unchosen vertex.

Objectives 1 and 2 can be modeled as a maximum vertex-weight minimum dominating set (MVWMDS) problem and formulated as integer linear programming (ILP) forms. The ILP formulation will be explained in Section III-C. After solving the MVWMDS problem, each vertex is determined to be either chosen or unchosen. Next, to achieve Objective 3, for every unchosen vertex, all edges connecting to its chosen adjacent vertices are searched and the one with the greatest edge weight is selected to approximate this unchosen vertex. Fig. 6 shows the result of the example in Fig. 5. A chosen vertex is represented in gray while an unchosen one is in white. An arrow pointing from an unchosen vertex p_i towards a chosen vertex p_j represents that the vertex p_i is approximated by the vertex p_j. The result from the filter similarity graph of the c^{th} input matrix can be represented by a set of arrows A_c, a set of chosen vertices SCV_c, and a set of unchosen vertices SUV_c. If there is an arrow pointing from the vertex p_u towards the vertex p_v, then $(u, v) \in A_c$. After the filter approximation, every FID_{mc} is updated by EQ (12). The filter dependency graph is then updated from Fig. 4(b) to Fig. 7. The reduction ratio of operation count is increased to $\left(1 - \frac{5}{15}\right) = 66.67\%$.

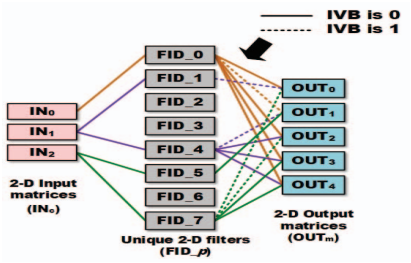


Fig. 7: The filter dependency graph corresponding to the example in Fig. 4 after filter approximation.

$$FID'_{mc} = \begin{cases} FID_{mc} & \text{, if } FID_{mc} \in SCV_c \\ v \text{ of } (FID_{mc}, v) \in A_c & \text{, if } FID_{mc} \in SUV_c \end{cases} \tag{12}$$

C. ILP Formulation

The ILP formulation of the MVWMDS problem in Section III-B4 is expressed as EQs (13) and (14). We use a two-round ILP algorithm. A filter similarity graph of the c^{th} input matrix is denoted as G_c. For each vertex p, there is a binary variable $x_p \in \{0, 1\}$. If x_p is 1, the vertex p is chosen; otherwise, the vertex p is unchosen.

First round of ILP:

$$\begin{aligned} \textbf{min} \quad & \sum_{p_i \in SID_c} x_{p_i}, \forall p_i \in SID_c \\ \textbf{s.t.} \quad & \left(x_{p_i} + \sum_{\substack{p_j \in SID_c \wedge \\ \{p_i, p_j\} \in E_c}} x_{p_j} \right) \geq 1, \forall p_i \in SID_c \end{aligned} \tag{13}$$

After solving the first round of ILP, the domination number of G_c, denoted as $\gamma(G_c)$, is obtained and will be used for the second round of ILP.

Second round of ILP:

$$\begin{aligned} \textbf{max} \quad & \sum_{p_i \in SID_c} x_{p_i} \times DI_{cp_i}, \forall p_i \in SID_c \\ \textbf{s.t.} \quad & \left(x_{p_i} + \sum_{\substack{p_j \in SID_c \wedge \\ \{p_i, p_j\} \in E_c}} x_{p_j} \right) \geq 1, \forall p_i \in SID_c \\ \textbf{s.t.} \quad & \sum_{p_i \in SID_c} x_{p_i} = \gamma(G_c), \forall p_i \in SID_c \end{aligned} \tag{14}$$

After solving the second round of ILP, each vertex can be determined to be either chosen or unchosen.

IV. Experimental Evaluation

We trained BNNs on the dataset CIFAR-10 [15] with BinaryNet [2]. Our network architecture is the same network used in [2], which contains six convolution layers. Our experiments were conducted on the last five convolution layers because the inputs of the first convolution layer are not binarized. For the filter approximation, we use 10K samples to evaluate DSIM. Keras [16] was used for training networks, and Gurobi [17] was used for our ILP solver.

Tables I shows the experimental results on CIFAR-10. The accuracy of CIFAR-10 after training with BinaryNet [2] is 88.74%. The reduction ratio of XNOR-popcount operations

TABLE I: Results of a BNN on CIFAR-10.

Layer (M × C) (E × F)			Reduction ratio (%) conv2 (128 × 128) (32 × 32)	conv3 (256 × 128) (16 × 16)	conv4 (256 × 256) (16 × 16)	conv5 (512 × 256) (8 × 8)	conv6 (512 × 512) (8 × 8)	Overall	Testing data accuracy (%)
Filter repetitions			36.33	52.89	48.46	63.13	60.16	**50.74**	**88.74**
Filter approximation	DSIMTH	0.90	37.59	56.86	52.13	63.79	60.22	52.57	88.82
		0.89	38.34	58.29	53.14	64.48	60.32	53.30	88.82
		0.88	39.03	59.79	54.17	65.90	60.48	54.13	88.73
		0.87	39.45	61.14	55.67	68.06	60.99	55.18	88.93
		0.86	40.56	63.11	57.10	70.47	61.55	56.50	88.47
		0.85	42.12	64.81	58.80	73.47	62.23	58.07	88.43
		0.84	43.04	66.87	60.68	76.77	63.12	**59.66**	**88.39**
		0.83	44.24	69.02	63.03	79.54	64.16	61.43	87.59
		0.82	45.67	71.40	65.29	83.06	65.11	63.32	87.41
		0.81	47.41	73.61	67.68	85.72	66.01	65.19	86.58
		0.80	50.02	75.65	70.07	87.48	67.21	67.22	85.76

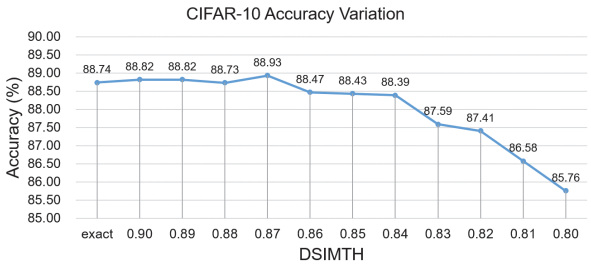


Fig. 8: Accuracy variation of BNNs with filter approximation CIFAR-10.

can reach 50.74% for CIFAR-10 by only applying filter repetitions. With filter approximation, the reduction ratio of the operations can further reach about 60% for CIFAR-10 with the accuracy loss of no more than 1%. Thus, the experimental results show that our approach can effectively reduce the operation count more than half without causing too much accuracy loss. Fig. 8 displays the accuracy variation over different values of DSIMTH. If the DSIMTH is not very low, the accuracy might fluctuate within a small range. We found that the accuracy of an approximated network might be higher than that of the originally trained network. We think this is because some mispredicted images might be correctly predicted after approximation. In general, the accuracy gradually drops as the DSIMTH becomes smaller. As the DSIMTH is smaller, it would take more time to conduct the filter approximation. The runtime of filter approximation with the DSIMTH of 0.8 is 296.1 seconds for CIFAR-10.

We conduct our experiments on 3×3 filters and our approach works for this filter size according to the experimental results. However, if the filter size becomes larger, the phenomenon of filter repetitions becomes rare. Nevertheless, most useful features in images are local, which means that the filter size prefers to be small in many applications. Among many choices of filter sizes, 3×3 filters in convolution layers are very popular and work well. Hence, our approach is still applicable and promising in practice.

V. Conclusions

In this paper, we propose a 2-D filter's convolutional result sharing approach for convolution layers in binary-weight-binary-input BNNs. We share the convolutional results by exploiting the filter repetitions and the filter approximation. The proposed scheme significantly reduces the number of required XNOR-popcount operations with accuracy loss of no more than 1%.

References

[1] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: training deep neural networks with binary weights during propagations," in Proc. NIPS, pp. 3123–3131, 2015.

[2] M. Courbariaux, Y. Bengio, "BinaryNet: training deep neural networks with weights and activations constrained to +1 or -1," ArXiv:1602.02830, 2016.

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

[4] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: imageNet classification using binary convolutional neural networks," in Proc. ECCV, pp. 525–542, 2016.

[5] P. Gysel, M. Motamedi, and S. Ghiasi, "Hardware-oriented approximation of convolutional neural networks," ArXiv:1604.03168, 2016.

[6] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-Net: training low bitwidth convolutional neural networks with low bitwidth gradients," ArXiv:1606.06160, 2016.

[7] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: training neural networks with low precision weights and activations," ArXiv:1609.07061, 2016.

[8] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in Proc. NIPS, pp. 1135-1143, 2015.

[9] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in Proc. CVPR, 2017.

[10] S. Han, H. Mao, and W. J. Dally, "Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding," ArXiv:1510.00149, 2015.

[11] W. Chen, J. T. Wilson, S. Tyree,K. Q. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in Proc. ICML, pp. 2285–2294, 2015.

[12] H. Kim, J. Sim, Y. Choi, L.-S. Kim, "A kernel decomposition architecture for binary-weight convolutional neural networks," in Proc. DAC, p. 60, 2017.

[13] C.-C. Chi and J.-H. R. Jiang, "Logic synthesis of binarized neural networks for efficient circuit implementation," in Proc. ICCAD, pp 84:1–84:7, 2018.

[14] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "FINN: A framework for fast, scalable binarized neural network inference," in Proc. Int. Symp. on Field-Programmable Gate Arrays, pp. 65–74, 2017.

[15] A. Krizhevsky, "Learning multiple layers of features from tiny images," MS thesis, University of Toronto, https://www.cs.toronto.edu/ kriz/cifar.html, 2009.

[16] "keras," https://keras.io/

[17] "GUROBI," https://www.gurobi.com/.