LETTER

Fast Inference of Binarized Convolutional Neural Networks Exploiting Max Pooling with Modified Block Structure

Ji-Hoon SHIN[†], *Nonmember and* **Tae-Hwan KIM**[†a)], *Member*

SUMMARY This letter presents a novel technique to achieve a fast inference of the binarized convolutional neural networks (BCNN). The proposed technique modifies the structure of the constituent blocks of the BCNN model so that the input elements for the max-pooling operation are binary. In this structure, if any of the input elements is +1, the result of the pooling can be produced immediately; the proposed technique eliminates such computations that are involved to obtain the remaining input elements, so as to reduce the inference time effectively. The proposed technique reduces the inference time by up to 34.11%, while maintaining the classification accuracy.

key words: *binarized neural networks, embedded systems, convolutional neural networks, inference, deep learning*

1. Introduction

Binarized convolutional neural network (BCNN) is a class of the CNN where each element of the weights and features is represented in a single bit. The inference of the full-precision CNN is usually involved with a massive number of multiply-and-accumulate (MAC) operations. In the inference of the BCNN, however, the MAC operation can be substituted by the simple XNOR-bitcount operation [1] equivalently, so that it is promising to achieve a fast inference even in the traditional CPU-based system. Moreover, the BCNN shows a good accuracy performance, which is comparable to that achieved by the full-precision CNN, in particular for the classification of the small-scale datasets (e.g. CIFAR-10 [2], SVHN [3]). For this reason, the BCNN is attracting practical interests to realize an artificial intelligence in a computation-limited device that has neither a GPU nor a dedicated accelerator [4].

The previous studies regarding the BCNN are focused on how to close the accuracy gap from the full-precision CNN while achieving an efficient implementation owing to the binarization. Courbariaux *et al.* presented BinaryNet [5], which is a pioneering work to show the potential of the BCNN that achieves a good accuracy in the classification of small-scale datasets. Rastegari *et al.* presented XNOR-Net [1], of which block structure is modified by employing the scaling factors to compensate the loss due to the binarization. Zhou *et al.* presented DoReFa-Net [6], where the gradients as well as the weights and features are represented as low-bitwidth data. Recently, Lin *et al.* presented ABC-Net [7], where the weights and features are binarized together with multiple binary bases, showing significant improvement in the classification accuracy. Some researchers developed dedicated processors to accelerate the inference of the BCNN [8], [9].

Manuscript received September 10, 2019.
Manuscript revised November 15, 2019.
Manuscript publicized December 3, 2019.

[†]The authors are with School of Electronics & Information Engineering, Korea Aerospace University, 76, Hanggongdaehak-ro, Deogyang-gu, Goyang-si, Gyeonggi-do, Republic of Korea.
a) E-mail: taehwan.kim@kau.ac.kr

DOI: 10.1587/transinf.2019EDL8165

This study shows an efficient technique to make the inference of the BCNN even faster. The proposed technique modifies the block structure so that the input elements for the max-pooling operation are binary. If any of the input elements is +1, the pooling operation can be finished early by returning +1 without considering other remaining input elements. The computations involved to obtain other remaining elements within the window can be eliminated effectively to reduce the inference time. The experimental results show that the proposed technique reduces the inference time by 21.76% and 34.11% for the CIFAR-10 and SVHN classification, respectively, while maintaining the classification accuracy.

The rest of this letter is organized as follows. Section 2 explains the conventional BCNN with its block structure. Section 3 presents the proposed technique to achieve a fast inference of BCNN. Section 4 evaluates the proposed technique based on the experimental results. Finally, Sect. 5 draws the conclusion.

2. Conventional BCNN

The BCNN model can be described as a set of blocks connected in series, where each block has such an identical structure that is shown in Fig. 1 [1], [5], [10]. In the figure, the input/output feature (**I**, **O**), and weight (**W**) for the block are binary. The pooling divides input into the non-overlapped pooling windows and select maximum within each window, extracting higher-level features. The scaling and batch normalization normalizes the input making it have a balanced distribution for the subsequent binarization. They are performed in a single step since they can be merged into an affine transform efficiently [1], [10]. The

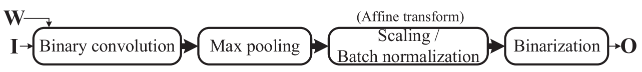

Fig. 1 Conventional structure of each block composing the BCNN [1], [5], [10]. The thick and thin arrows represent the real and binary elements, respectively.

Algorithm 1 Processing flow of the proposed block (for the inference), where the max-pooling is performed with the window size of $k \times k$ and the stride of s.

```
Require: input feature $\mathbf{I} \in \mathcal{B}^{w_i \times h_i \times c_i}$ and weight $\mathbf{W} \in \mathcal{B}^{w_f \times h_f \times c_i}$.
Ensure: output feature $\mathbf{O} \in \mathcal{B}^{w_o \times h_o}$.
 1: for each $(x, y)$, where $x \in \{1, 2, \cdots, w_o\}$ and $y \in \{1, 2, \cdots, h_o\}$ do
 2:     $result \leftarrow -1$, where $result$ denotes the result of a max-pooling operation.
 3:     for each $(m, n)$, where $m \in \{1, 2, \cdots, k\}$ and $n \in \{1, 2, \cdots, k\}$ do    ▷ iteration for the elements within the max-pooling window.
 4:         $result \leftarrow sign(\alpha \cdot (\mathbf{I}_\Psi \odot \mathbf{W}) + \beta)$, where $\mathbf{I}_\Psi$ denotes the receptive field in $\mathbf{I}$ of the size $w_f \times h_f \times c_i$ centering around $(s(x-1)+m, s(y-1)+n)$. ▷ affine transform and binarization of the binary dot-product result.
 5:         if $result = +1$ then
 6:             break the inner loop.    ▷ Early terminate the iteration for the elements within the pooling window.
 7:         end if
 8:     end for
 9:     $O(x, y) \leftarrow result$
10: end for
```

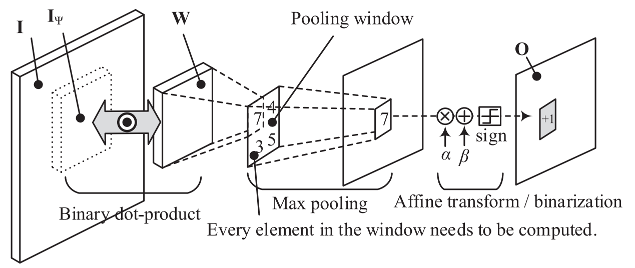

Fig. 2 Processing in the conventional BCNN block, where the size of the pooling window is 2×2.

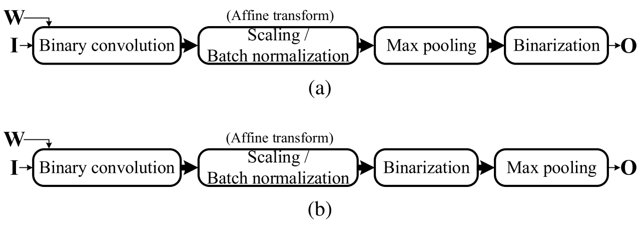

Fig. 3 Modified structures of each block composing the BCNN, where (a) and (b) are used for the training and inference, respectively. The thick and thin arrows represent the real and binarized elements, respectively.

block structure shown in the figure is quite different from that of the full-precision CNN model [4], in that the pooling is performed just after the binary convolution so as to avoid the information loss due to the binarization.

The processing in the block is illustrated in Fig. 2. As shown in the figure, each element in the pooling window is calculated by the binary dot-product between the weights and receptive field in the input feature, where the receptive field is denoted by $\mathbf{I}_\Psi$ in the figure and $\odot$ is the binary dot-product operator. The maximum of the elements in the pooling window is selected, transformed by the affine function, and binarized into ±1 by the sign function. For the affine transform, the scaling parameter α and shifting parameter β can be obtained by investigating the statistics of the weights and features while performing the training. It should be noted that every element in the pooling window needs to be considered for the pooling operation as long as its value is less than the maximum possible real value.

3. Proposed Technique

The proposed technique modifies the block structure in order to achieve a fast inference. The processing steps in each block are rearranged so that the scaling and batch normalization are performed prior to the pooling, as shown in Fig. 3 (a). The structure is modified again by interchanging the pooling with the binarization, as shown in Fig. 3 (b). Because this can be considered to applying the binarization, which is a *monotonically-increasing* function, to each element in the pooling window in a distributive way, it makes no difference in propagating the feature forward. However in the backward propagation, the block structure shown in Fig. 3 (b) cannot be used successfully. This is because it is much probable that there are more than one maximum elements within the pooling window as each element is binarized; one of which is to be selected ambiguously to propagate the gradient backward. In consequence, the proposed technique uses the block shown in Fig. 3 (b) for the purpose of the inference, with the weights which have been trained for the block shown in Fig. 3 (a).

The modified block can make the inference fast. The elements within the pooling window has been binarized; hence, the max-pooling operation can be finished early by returning +1 if any of the elements is found to be +1 because the maximum possible value of the binarized elements is obviously +1. Algorithm 1 delineates the processing flow of the modified block, where the width, height, and channel are denoted by w, h, and c, respectively, with the appropriate subscripts (i: input feature, o: output feature, f: weight), and $\mathcal{B} \triangleq \{-1, +1\}$. As described in Line 6, the iteration for the elements within the pooling window is terminated as soon as any of the elements is found to be +1. As Algorithm 1 describes the per-channel processing flow, the output feature of multiple channels may be produced by performing it repetitively with the weight of each channel.

Figure 4 illustrates the processing in the modified block in the proposed technique. In the figure, the result of the pooling operation can be obtained immediately since one of the elements within the pooling window is found to be +1; the operations to compute other remaining elements are re-

Table 1 Inference time of the BCNN model for the CIFAR-10 classification.

Block	Input Feature Size (bits)	Weight Size (bits)	Pooling Size[a]	Pooling Stride[b]	Conv. Processing Time[c] (ms)	Prop. Processing Time[c] (ms)	Reduction (%)
CB1	$32 \times 32 \times 24$	$3 \times 3 \times 24 \times 128$	-	-	72.948	74.183	-
CB2	$32 \times 32 \times 128$	$3 \times 3 \times 128 \times 128$	2	2	184.225	134.516	26.98
CB3	$16 \times 16 \times 128$	$3 \times 3 \times 128 \times 256$	-	-	93.226	94.318	-
CB4	$16 \times 16 \times 256$	$3 \times 3 \times 256 \times 256$	2	2	167.445	113.171	32.41
CB5	$8 \times 8 \times 256$	$3 \times 3 \times 256 \times 512$	-	-	84.629	86.133	-
CB6	$8 \times 8 \times 512$	$3 \times 3 \times 512 \times 512$	2	2	147.378	80.876	45.12
FCB1	8192	8192×1024	-	-	13.581	13.711	-
FCB2	1024	1024×1024	-	-	1.746	1.778	-
FCB3	1024	1024×10	-	-	0.024	0.025	-
Overall	-	-	-	-	765.202	598.711	21.76

[a] Corresponding to k in Algorithm 1.
[b] Corresponding to s in Algorithm 1.
[c] Processing time measured for the models designed with and without the proposed technique.

Table 2 Inference time of the BCNN model for the SVHN classification.

Block	Input Feature Size (bits)	Weight Size (bits)	Pooling Size[a]	Pooling Stride[b]	Conv. Processing Time[c] (ms)	Prop. Processing Time[c] (ms)	Reduction (%)
CB1	$32 \times 32 \times 24$	$5 \times 5 \times 24 \times 128$	2	2	159.246	76.044	52.25
CB2	$16 \times 16 \times 128$	$3 \times 3 \times 128 \times 256$	-	-	93.292	94.468	-
CB3	$16 \times 16 \times 256$	$3 \times 3 \times 256 \times 256$	-	-	180.473	183.087	-
CB4	$16 \times 16 \times 256$	$3 \times 3 \times 256 \times 256$	4	4	162.671	36.460	77.59
CB5	$4 \times 4 \times 256$	$3 \times 3 \times 256 \times 128$	-	-	4.208	4.285	-
CB6	$4 \times 4 \times 128$	$3 \times 3 \times 128 \times 128$	-	-	2.259	2.278	-
FCB1	2048	2048×128	-	-	0.388	0.394	-
FCB2	128	128×128	-	-	0.046	0.046	-
FCB3	128	128×10	-	-	0.007	0.008	-
Overall	-	-	-	-	602.590	397.070	34.11

[a] Corresponding to k in Algorithm 1.
[b] Corresponding to s in Algorithm 1.
[c] Processing time measured for the models designed with and without the proposed technique.

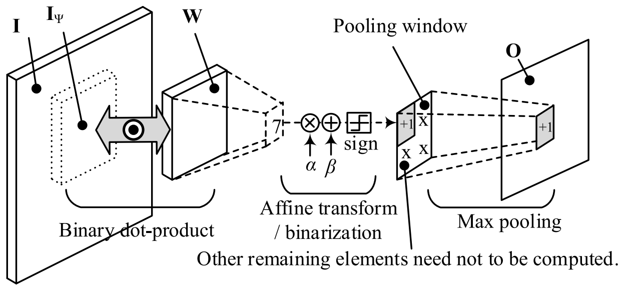
I
I_Ψ
W
Pooling window
O
+1
x
x
x
+1
sign
α
β
Binary dot-product
Affine transform / binarization
Max pooling
Other remaining elements need not to be computed.

Fig. 4 Processing in the modified BCNN block used for inference in the proposed technique, where the size of the pooling window is 2×2.

dundant and thus can be eliminated effectively. This is contrast to the conventional processing in Fig. 2, where every element within the window is always computed to get the result of the pooling operation.

The ratios of ± 1's within the pooling windows affect the speed-up by the proposed technique. To be specific, if the ratios of -1's within the pooling windows were so high, the speed-up by the proposed technique might not be significant. The proposed technique avoids such situation effectively by using the modified block shown in Fig. 3 (a) for training a model. Since the batch normalization provides elements with a balanced distribution, it is not probable that the ratio of -1's within the pooling window is so high. Such modification does not have any noticeable effect on the classification accuracy while making the speed-up by the proposed technique significant, and this will be validated based on the experimental results in the next section.

4. Experimental Results

The efficacy of the proposed technique has been evaluated by investigating the inference time and accuracy of the image classification tasks. The classification tasks of two data sets (CIFAR-10 [2] and SVHN [3]) have been performed based on the BCNN models whose structures are summarized in Tables 1–2. Each BCNN model is composed of sev-

Table 3 Top-1 classification accuracy.

Classification Task	Conv. BCNN[a]	Prop. BCNN[a]	Full-Precision CNN
CIFAR-10	88.89%	88.88%	91.77%
SVHN	95.31%	95.40%	97.40%

[a] BCNN models designed with and without the proposed technique.

eral convolutional blocks (CB) and fully-connected blocks (FCB). Some CBs contain the pooling operations and others do not. The BCNN model for the CIFAR-10 classification has the same structure of that presented in [5] and the BCNN model for the SVHN classification has been slightly modified taking the binarization into account from that presented in [6].

The BCNN model for the CIFAR-10 classification has been trained for 250 epochs based on the stochastic gradient descent optimizer of the initial learning rate of 0.1 and momentum of 0.9, with the batch size of 512. The BCNN model for the SVHN classification has been trained for 400 epochs with the same optimizer and batch size setting. Any pre-processing or data augmentation techniques that can affect the classification accuracy have not been used. Table 3 summarizes the classification accuracy achieved by the BCNN models with and without the proposed technique along with that achieved by the full-precision models. The table manifests that the BCNN is viable to achieve a good accuracy for the classification of such small-scale datasets, and this is as high as that achieved by the full-precision model. More importantly, the difference of the accuracy caused by the modified block structure in the proposed technique is not noticeable.

The classification tasks based on the BCNN models have been implemented using C language and the inference time has been measured under the embedded system with 800MHz ARM Cortex A9 processor, 1GB SDRAM, and no GPU. Any SIMD optimization has not been considered in the implementation. Tables 1–2 analyze the inference time by showing the per-block processing time. As shown in the tables, the proposed technique is effective to reduce the processing time of a block that contains the pooling operation. The processing time of a block that does not contain the pooling operation has been increased because of the overhead to check the condition to terminate the iteration, but it is slight. As a result, the proposed technique reduces the overall inference time by 21.76% and 34.11%, for the CIFAR-10 and SVHN classification, respectively.

It is worth making additional remarks:

- Fast inference is practically useful particularly for realizing near real-time applications such as the object detection and semantic segmentation for the autonomous driving. In addition, fast inference may lead low energy consumption if the power consumption is maintained.
- There is a ready-made framework that makes the inference faster targeting embedded devices [11]. It is mainly based on the machine-dependent optimization; whereas the proposed technique is based on the modification of the processing structure of the BCNN. Furthermore, the framework is based on the post-training quantization of the full-precision model. This is quite different from the approach of the BCNN, where the models are developed so that each element of features and weights is represented in one bit and trained considering the binarization effects [1], [5]–[7]. More importantly, the proposed technique is orthogonal to the framework; that is, it is viable to make the inference faster in combination with the machine-dependent optimizations provided by the framework.

5. Conclusion

This letter presents an efficient technique to make the inference of the BCNN fast. The proposed technique modifies the structure of the constituent blocks of the BCNN so that the max-pooling operation may be finished early without considering every element within the pooling window. When applying the proposed technique to the BCNN models for the CIFAR-10 and SVHN classification, the overall inference time of the BCNN is reduced by 21.76% and 34.11%, respectively, while maintaining the classification accuracy.

Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2018R1D1A1A09082763) and the GRRC program of Gyeonggi province [2017-B06, Research of Sensor Data Processing Systems for Smart Mobility Assistance]. The tools and environments were supported by IDEC, Korea.

References

[1] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," Proc. European Conf. Computer Vision, pp.525–542, Springer, March 2016.
[2] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Tech. Rep., Citeseer, 2009.
[3] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A.Y. Ng, "Reading digits in natural images with unsupervised feature learning," Proc. NIPS Workshop on Deep Learning and Unsupervised Feature Learning, pp.1–9, NeurIPS Foundation, Dec. 2011.
[4] V. Sze, Y.-H. Chen, T.-J. Yang, and J.S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," Proc. IEEE, vol.105, no.12, pp.2295–2329, Dec. 2017.
[5] M. Courbariaux and Y. Bengio, "BinaryNet: training deep neural networks with weights and activations constrained to +1 or −1," arXiv: 1602.02830, 2017.
[6] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-net: Training low bitwidth convolu-tional neural networks with low bitwidth gradients," arXiv preprint arXiv:1606.06160, 2016.
[7] X. Lin, C. Zhao, and W. Pan, "Towards accurate binary convolutional neural network," Proc. Advances in Neural Information Processing Systems, pp.345–353, Dec. 2017.

[8] 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," Proc. International Symp. Field-Programmable Gate Arrays, pp.65–74, ACM, 2017.

[9] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "YodaNN: An architecture for ultralow power binary-weight cnn acceleration," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol.37, no.1, pp.48–60, Jan. 2017.

[10] Y. Wang, J. Lin, and Z. Wang, "An energy-efficient architecture for binary weight convolutional neural networks," IEEE Trans. VLSI Syst., vol.26, no.2, pp.280–293, Feb. 2018.

[11] "TensorFlow Lite," 2018. Software available from tensorflow.org.