Functionally-Predefined Kernel: a Way to Reduce CNN Computation

Yuta INOUCHI*, Hayato YAMAKI†, Shinobu MIWA†, and Tomoaki TSUMURA*

*Nagoya Institute of Technology, Gokiso, Showa, Nagoya, Japan
Email: camp@matlab.nitech.ac.jp
†The University of Electro-Communications, 1-5-1 Chofugaoka, Chofu, Tokyo, Japan

***Abstract*—Convolutional Neural Networks (CNNs) have achieved high classification accuracy in image recognition, and now, they are widely used for numerous applications. For higher accuracy or more advanced applications, CNNs need to consume tremendous computational resources and time. Hence, many studies for reducing the computational cost of CNNs are actively being conducted. However, many previous methods for reducing the computational cost lead to a non-negligible loss in output accuracy. Therefore, it is still a challenge to reduce the computational cost of CNNs with keeping the output accuracy high. In this paper, we propose a novel concept "Functionally-Predefined Kernel" to reduce the computational cost for CNN training and discuss the potential of computation reuse to reduce the computational cost for CNN inference. Our experimental results show that the number of parameters to be trained can be significantly reduced by utilizing Functionally-Predefined Kernels without accuracy loss. In addition, we revealed that CNN's inference process includes many convolution operations with the same inputs and computation reuse, therefore, has high affinity to CNN computation.**

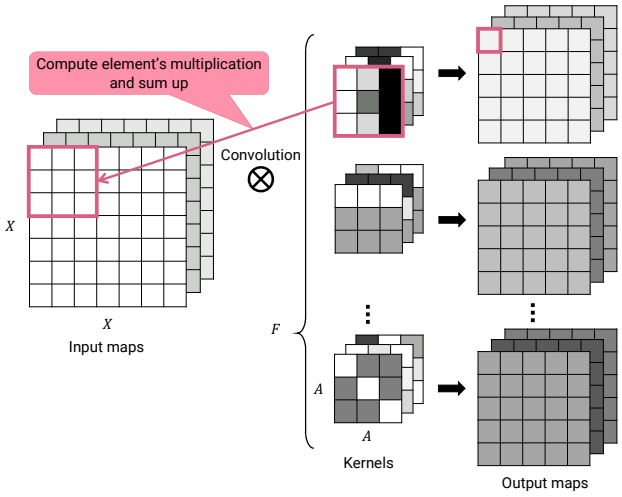

Fig. 1. Operations in a convolutional layer

I. Introduction

Convolutional Neural Networks (CNNs) have achieved high classification accuracy in image recognition, and now they are widely used for numerous applications such as face recognition, character recognition, and pedestrian detection in automated driving. CNNs significantly improve the recognition accuracy as they scale in both width and depth; consequently, both execution time and energy required for CNN computation, especially for the training process, which needs a considerable amount of computational resources and time in general, are increasing. To address this problem, many techniques that approximate CNN computation have been developed to reduce the computational cost. However, depending on the degree of approximation, these approaches drop some critical information included in the original CNN, thereby showing the significant reduction in the recognition accuracy, which is intolerable to some classes of applications.

In this paper, we aim to improve both training and inference time for CNNs while maintaining the recognition accuracy. To reduce the training cost, we propose a new concept called Functionally-Predefined Kernel, which reduces learnable parameters in CNN by exploiting a set of primal kernels needed for feature detection. Additionally, combining Functionally-Predefined Kernel with computation reuse can reduce the inference cost. Our experimental results show that Functionally-Predefined Kernel has an excellent potential to reduce both training and inference cost of CNNs.

II. Convolutional Neural Networks

CNN is a neural network that has multiple structurally-unique layers called convolutional layers. A convolutional layer has multiple two-dimensional parameter arrays called *kernels*. Each convolutional layer receives multiple two-dimensional images called *feature maps* as its input, and then outputs multiple new feature maps generated by convolving the input feature maps with the kernels. CNN sequentially performs this step for a given input image with the help of stacked convolutional layers, allowing to achieve high recognition accuracy.

Each input feature map is convolved with multiple kernels. Convolution is an operation that computes element-wise multiplication of an input feature map and a kernel and then sums up the result into an output feature map. Figure 1 shows operations performed in a convolutional layer with multiple kernels. As shown in the figure, the convolutional layer generates F output feature maps by convolving an input feature map of $X \times X$ with F kernels of $A \times A$ in each.

In this example, the element located in the upper left of the top output feature map is calculated by convolving the upper left 3×3 region in the input feature map with the top kernel. The other elements in the output feature map are computed by repeatedly moving the kernel over the input feature map and performing the above operation.

Values of kernels used in convolutional layers are determined through training. In a training process, kernel parameters are repeatedly updated for a set of training data based on errors between CNN's and correct outputs.

Both training and inference perform a large number of product-sum operations in the convolutional layers, and reducing this computational cost is, therefore, an important issue. To make matters worse, in recent years, the numbers of layers and kernels within one CNN tend to increase in order to improve recognition accuracy, thereby increasing computation amount and data required for training and inference. For example, the numbers of parameters used and product-sum operations performed in the convolution layers included in AlexNet [1], which was proposed in 2012, are 2.3M and 666M, respectively, while VGG-16 [2], which was proposed in 2014, has 14.7M parameters and 15.3G product-sum operations. Moreover, training ResNet [3], which was proposed in 2015 and has 152 convolutional layers, for ImageNet [4] spent several weeks even on a computer equipped with multiple GPUs. As another example, VGG-16 takes 10.67ms [5] to calculate the output for one input image using GPUs, while AlexNet takes only 0.54ms. This increase in computation time is an obstacle to using CNNs, especially for real-time applications that need to be executed under severe timing and resource constraints. Therefore, it is required for CNNs to calculate their outputs in a short time while keeping the recognition accuracy.

III. Related Work

Many studies have been done for accelerating training and inference in CNNs. We summarize them in this section.

A. Parameter Quantization

One approach to reducing the computational cost of CNN is to quantize the parameters. For example, in XNOR-NET[6], each multiplication performed through training and inference is replaced with a simple XNOR operation by limiting input values and weight parameters to the binary values of $+1$ and -1. The authors of XNOR-NET reported that parameter quantization can reduce parameter size and inference time by $1/32$ and $1/52$ for a CNN, respectively. As another approach to quantizing parameters, it has been proposed to replace each multiplication with a shift operation by expressing weights as binary logarithms [7].

While parameter quantization reduces the computation cost of CNNs, it rounds trained parameters to a small set of low-precision values depending on quantization policies. As a result, CNNs with parameter quantization often show a significant drop in the recognition accuracy, which is not negligible for some applications.

B. Hardware Acceleration

Use of hardware accelerators is beneficial to reducing the computation time of CNNs. One example is DianNao [8], a many-core accelerator that performs high-speed inference for large-scale CNNs. A core consists of an input and output buffers, a weight buffer, and an NFU (Neural Functional Unit) equipped with many product-sum operation units. Input/output feature maps and kernels included in CNN are stored into the three on-chip buffers to reduce the data movement cost. Furthermore, the NFU performs many product-sum operations in parallel for high-speed inference. However, this approach needs a number of cores and product-sum operation units to quickly perform many convolution operations required for large-scale CNNs, resulting in an increase in circuit area and power consumption.

C. Parameter Clustering

We proposed kernel clustering to reduce the computational cost of CNNs [9]. Our approach is based on the observation that many kernels trained with different CNN architecture and/or datasets often have similar values. This nature motivates us to cluster kernels included in a trained CNN for reducing the computational cost. More specifically, computation at convolutional layers can be commonized for each cluster by replacing each kernel in a cluster with the representative kernel in the cluster. Since this commonized computation can be performed in many CNNs, we developed an accelerator that employs hardware specialized for the above computation.

Throughout this study, we found that *kernels with similar functions exist among various CNNs* and then foresee that *functions that extract simple features such as edges and blobs would be included in any trained CNN,* producing an idea that we can predefine some kernels.

D. Reduction in Learnable Parameters

Shrinking learnable parameters within CNNs is a practical approach to reduction in the training cost. Based on this direction, Juefei-Xu et al. proposed Local Binary Convolution (LBC) [10], which enables to replace convolutional layers with LBC layers. Networks obtained through the replacement are called Local Binary Convolutional Neural Networks (LBCNNs). An LBC layer comprises of some fixed sparse predefined binary convolutional kernels, a non-linear activation function, and a set of 1×1 kernels. While the 1×1 kernels are learnable (i.e., updated during a training process of LBCNN), the binary convolutional kernels are not. Therefore, the LBC layer significantly saves the number of learnable parameters when compared to the original convolutional layer.

Values of binary convolutional kernels are determined based on Local Binary Patterns (LBPs). LBP is a simple yet powerful descriptor derived from the face recognition field and is widely used in pattern recognition and image processing applications. The method to compute LBP features is shown in Fig. 2. When calculating LBP features, first a part of the image is cut out as a patch, as shown on the left and center in Fig. 2, and then the luminance of the central pixel in the patch is

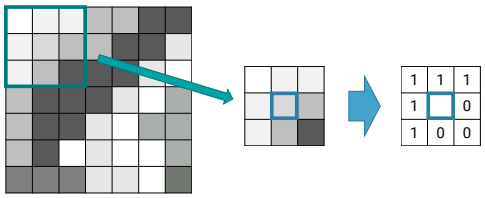

Fig. 2. LBP features

compared with that of the neighboring pixels. The computation result is set to 1 if the luminance of a neighboring pixel is greater than that of the central pixel, while it is set to 0 if not, as shown on the right in Fig. 2. Finally, a bit string of the computation results is read sequentially and mapped to a decimal number as the LBP feature value of the central pixel. Based on this idea, the LBCNN's approach creates a kernel where each parameter is set to 0 or ± 1 randomly selected according to the sparsity specified. This randomly generated kernel is used as a predefined kernel in the LBC layer.

LBCNNs have fewer learnable parameters than conventional CNNs but show the recognition accuracy comparable to the conventional CNNs in most cases. The authors of LBCNN demonstrate, both theoretically and empirically, that LBC layers are good approximation of convolutional layers. Moreover, the authors show that LBCNN achieves the almost same recognition accuracy as conventional CNNs for MNIST, SVHN, CIFAR-10, and ImageNet datasets while reducing the computational cost significantly.

We move forward with this idea as Functionally-Predefined Kernel, which uses simple but meaningful kernels to extract typical features such as edges and blobs, instead of randomly generated kernels. We consider that the use of Functionally-Predefined Kernel leads to a further reduction in the training cost of CNNs while maintaining the recognition accuracy when compared to LBCNN. In addition, Functionally-Predefined Kernel can reduce the inference cost of CNNs with the help of computation reuse, because many multiplications performed in convolution operations have the same input due to the binary parameters used in Functionally-Predefined Kernels, as described in Sec. IV-B. Thus, Functionally-Predefined Kernel is a more promising approach to reducing the computational cost of CNNs.

IV. Functionally-Predefined Kernel

In this section, we introduce the concept of Functionally-Predefined Kernel and how to reduce the kernel parameters to be trained.

A. Outline

In respect of organic brains, it is known that a part of the primary visual cortex has a function to respond to and recognize an edge in a specific direction. On the other hand, in well-trained CNNs, it is generally known that simple features are extracted in the former convolutional layers, and more abstracted information is extracted in the latter layers [11]. Practically, a well-trained CNN often has kernels that seem

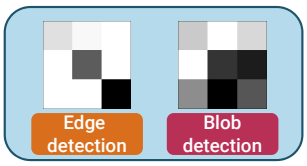

Fig. 3. Typical kernels in former layers.

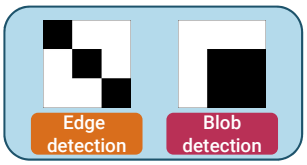

Fig. 4. Kernels specialized for extracting simple features

to be in charge of simple feature detection such as edges or blobs (Fig. 3), in former layers.

Assuming that such kernels are actually trained to undertake edge- or blob-detection, any CNN application may need to obtain such kernels by time-consuming CNN training, and pre-trained or pre-defined fixed kernels for such simple feature detection can contribute to low computation cost for CNN training. In this paper, we define portable kernels that are specialized to detect simple features (Fig. 4) and call them *Functionally-Predefined Kernels*. The parameters in Functionally-Predefined Kernels are permanently fixed, and not updated through CNN training.

Functionally-Predefined Kernel reduces the training cost of CNNs as follows. CNN usually has to do a complicated and time-consuming process to adjust the parameters to the optimal values. More specifically, it, also known as a gradient descent method, needs to repeatedly compute a partial derivative of a given loss function with respect to each parameter for a given training dataset. We can skip this process for fixed parameters when training CNNs with Functionally-Predefined Kernels so that the training cost can be reduced.

B. Experimental Setup

We evaluated the effect of Functionally-Predefined Kernel on recognition accuracy and learnable parameter count of CNNs. We use Chainer [12] as a deep learning framework, and train two CNNs with MNIST [13] and CIFAR-10 [14], respectively. Figure 5 shows our CNN architecture trained with the MNIST dataset, while Fig. 6 shows our CNN architecture trained with the CIFAR-10 dataset.

We use the 25 patterns shown in Fig. 7 as Functionally-Predefined Kernels. These kernels are determined based on Higher-order Local Auto-Correlation (HLAC) [15], which is a statistical feature for image recognition and satisfies position invariance and additivity. The practicality of HLAC is demonstrated with some applications such as object detection[16] and gesture recognition[17]. We embed these 25 kernels in the first convolutional layer within each CNN and then fix these parameters through training.

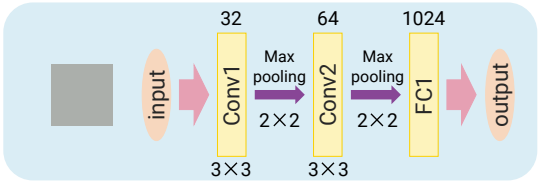

Fig. 5. CNN architecture for MNIST dataset

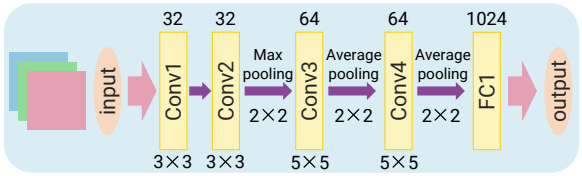

Fig. 6. CNN architecture for CIFAR-10 dataset

Since mask patterns used in HLAC are represented with binary values of 0 and 1, we use two values for the elements of Functionally-Predefined Kernels. Here, initial values randomly generated by Chainer follow a Gaussian distribution that has a mean of 0 and a variance of $1/\sqrt{fan\_in}$ (fan_in: number of inputs) so that about 99.7% of these values are in the range of $[-3/\sqrt{fan\_in},\ 3/\sqrt{fan\_in}\ ]$ theoretically. Therefore, we decided to use $\pm 3/\sqrt{fan\_in}$ as the element values of Functionally-Predefined Kernels. In Fig. 7, black elements represent $3/\sqrt{fan\_in}$, while white elements represent $-3/\sqrt{fan\_in}$.

C. Experimental Results

Figures 8 and 9 show the recognition accuracy of the CNNs. Figure 8 represents the case that the CNN shown in Fig. 5 is trained with the MNIST dataset, while Fig. 9 represents the case that the CNN shown in Fig. 6 is trained with the CIFAR-10 dataset. The vertical axes represent recognition accuracy, while the horizontal axes represent the number of epochs. The two orange lines shown in each figure represent the CNN with Functionally-Predefined Kernel, while the two green lines represent the CNN without Functionally-Predefined Kernel. The solid and dashed lines represent Top-1 and Top-3 accuracy, respectively.

Both figures show that the recognition accuracy of the CNNs with Functionally-Predefined Kernel is slightly lower that of the CNNs without Functionally-Predefined Kernel at the end of training. We observed a decrease of only 0.08% and 1.57% in Top-1 recognition accuracy for MNIST and CIFAR-10, respectively, while 0.002% and 0.42% in Top-3.

TABLE I shows the number of learnable parameters in the first layer of the CNN shown in Fig. 5 with and without Functionally-Predefined Kernel. The table shows that Functionally-Predefined Kernel can reduce the learnable parameter count by about 78% (i.e., from 864 to 189). From these results, though our experiment is still preliminary, we think Functionally-Predefined Kernel is effective to reduce the computational cost of CNNs while keeping the recognition accuracy.

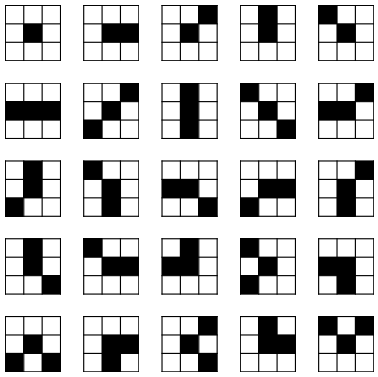

Fig. 7. Functionally-Predefined Kernels based on HLAC

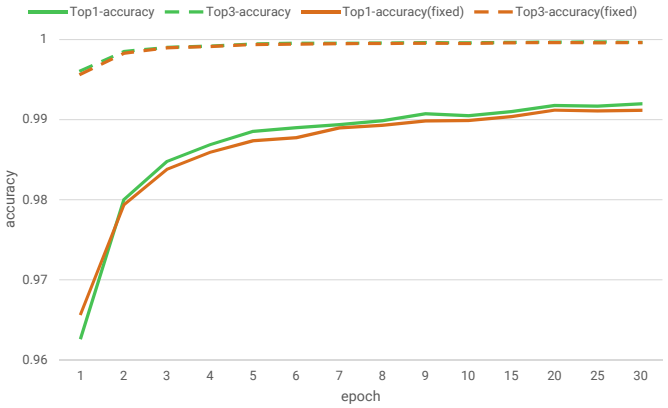

Fig. 8. Accuracy (MNIST dataset)

V. Cooperating Functionally-Predefined Kernels with Computation Reuse

In this section, we describe the affinity of computation reuse to the inference processing and estimate how much the amount of computation needed for inference can be reduced by utilizing computation reuse together with Functionally-Predefined Kernels.

A. Computation Reuse for Inference

Computation reuse is a technique that accelerates applications by omitting redundant computation. It stores a pair of an input sequence and an output of a computation block, such as a function, into a buffer and avoids re-compututation for the block by reusing the preserved output when the current input sequence matches the preserved input sequence. Computation reuse was initially developed as a programming technique that accelerates frequently-executed functions such as recursive functions in functional languages, but now it extends to hardware. There exist various granularities of computation reuse at hardware level, i.e., from instruction to function and loop level.

An inference process of CNN is an application well-suited for computation reuse. In the field of image processing, pixels located nearby each other show similar values in general; therefore, in an input feature map of CNN, there are

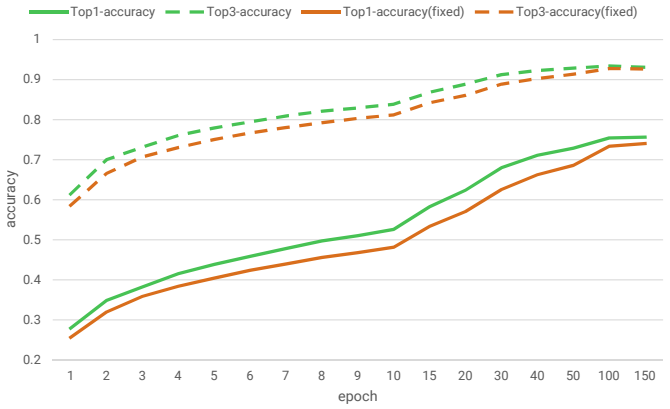

Fig. 9. Accuracy (CIFAR-10 dataset)

TABLE I
LEARNABLE PARAMETER COUNT IN THE FIRST CONVOLUTIONAL LAYER

CNN	Learnable parameter count
CNN w/o Functionally-Predefined Kernel	864
CNN w/ Functionally-Predefined Kernel	189

many elements that have similar values. In addition, when Functionally-Predefined Kernel is applied to CNN, kernels can be represented as the combinations of the limited number of values ($\pm 3/\sqrt{fan\_in}$). This means that a convolution operation possibly includes many multiplications that have the same combination of a multiplier and a multiplicand.

One example in this direction is the study conducted by Jiao et al., computation reuse with approximate matching for multiplication in CNN computation [18]. This approach statically analyzes a trained CNN to find multiplication patterns frequently-executed during convolution operations and records pairs of their inputs and outputs. With this record, it removes redundant multiplication included in the inference process of the CNN by approximately matching the inputs of a reuse-targeted multiplication with the recorded inputs. More specifically, only the upper nine bits of input values, or only the exponent field bits of 32-bit floating point, are used for the comparison. If both inputs of a targeted multiplication are approximately equal to those of the recorded inputs, the execution of the multiplication can be omitted with the corresponding recorded output. The authors report that it can save 80% of the overall multiplications in the inference for the MNIST dataset.

Similar to this approach, Functionally-Predefined Kernel limits input patterns for multiplications included in convolution operations and therefore increases an opportunity to reuse the computation results. One advantage of Functionally-Predefined Kernel over the previous work is that we can optionally use complete matching instead of approximate matching to check the availability of computation reuse. Since Functionally-Predefined Kernels have binary values, convolution operations included in CNNs with Functionally-Predefined Kernels are more likely to perform multiplications

TABLE II
NUMBER OF COMBINATIONS OF MULTIPLIER AND MULTIPLICAND (BASELINE)

Number of appearances	Quantization degree		
	fixed8	float16	float32
1–5	14215.2	96793.6	102700.0
6–10	8101.7	28584.9	29299.1
11–50	16622.8	15218.4	14451.2
51–100	2453.2	546.7	540.7
101–500	686.2	241.2	233.6
501–1000	14.9	0.0	0.0
1001–5000	0.0	0.0	0.0
5001–10000	0.0	0.0	0.0
10000+	0.0	0.0	0.0
Total	42094.0	141384.8	147224.6

TABLE III
NUMBER OF COMBINATIONS OF MULTIPLIER AND MULTIPLICAND (W/ FUNCTIONALLY-PREDEFINED KERNEL)

Number of appearances	Quantization degree		
	fixed8	float16	float32
1–5	11958.5	22390.0	22659.7
6–10	5698.2	8117.4	8155.7
11–50	5017.7	3269.6	3225.7
51–100	71.1	11.4	11.0
101–500	91.9	89.7	89.7
501–1000	102.4	102.1	102.1
1001–5000	227.8	228.0	228.0
5001–10000	4.9	4.9	4.9
10000+	0.3	0.3	0.3
Total	23172.8	34213.4	34477.1

that have the same combinations of a multiplier and a multiplicand even if values of feature maps are not quantized. Therefore, computation reuse combined with Functionally-Predefined Kernel will be able to achieve higher recognition accuracy than computation reuse combined only with parameter quantization.

B. Experimental Setup

We investigated how many multiplications that have the same combinations of a multiplier and a multiplicand are included in the convolution operations conducted in the first layer of the CNN shown in Fig. 6. The first convolutional layer needs to perform 777,600 multiplications for a given input image, assuming 32 nodes, three channels in each node, 30×30 convolution operations in each channel, and 3×3 multiplications performed in one convolution. As described in Sec. III-A, parameter quantization is often used to reduce the computation cost of CNNs. Therefore, we investigated the case that all parameters except for Functionally-Predefined Kernels are quantized with 16-bit floating point and 8-bit fixed point, in addition to the case that the parameters are expressed by 32-bit floating point, which is the standard precision used in Chainer.

C. Experimental Results

TABLES II and III show our analysis results. These tables show the number of combinations of a multiplier and a multiplicand for each number of multiplications executed through

the convolution process. The right three columns represent the combination count when varying precision of values. Fixed8, float16, float32 represent 8-bit fixed point, 16-bit floating point and 32-bit floating point, respectively. We summarize the combination count averaged over ten input images randomly selected from the CIFAR-10 dataset in the figure. For example, TABLE II shows that 540.7 combinations of a multiplier and a multiplicand are executed 51–100 times on average when using float32 (see the 6th row in the rightmost column).

TABLE II shows that the multiplications that both have the same combinations of a multiplier and a multiplicand and are frequently executed are still limited, though using fixed8 increases the number of such multiplications. For example, the number of combinations of a multiplier and a multiplicand that are executed 500+ times is only 14.9 even in case of fixed8. This means that we have a little chance to reuse results of multiplications for our CNN when using computation reuse combined with parameter quantization.

On the other hand, TABLE III shows that the CNN with Functionally-Predefined Kernel includes much more multiplications that have the same combinations of a multiplier and a multiplicand. Since some combinations appear several thousand to ten thousand times, we can expect that computation reuse combined with Functionally-Predefined Kernel works more effectively. For example, in case that neither Functionally-Predefined Kernel nor parameter quantization are used, the number of multiplications can be reduced to about 19% ($= 147224/777600$) by computation reuse because there are approximately $147{,}224$ unique patterns of multiplications. When applying Functionally-Predefined Kernel to the CNN, this number can be reduced to about 4.4%. Furthermore, we can reduce the number of unique multiplications to about 3.0% when using both Functionally-Predefined Kernel and fixed8. These results show that the CNN with Functionally-Predefined Kernel has many chances of computation reuse even without parameter quantization. It is possible that computation reuse combined with Functionally-Predefined Kernel can drastically reduce the inference cost of CNNs while avoiding the degradation of the recognition accuracy.

VI. Conclusions

In this paper, we proposed Functionally-Predefined Kernel to reduce the training cost of CNNs and discussed the effectiveness of combining Functionally-Predefined Kernel with computation reuse for improvement in inference performance. Our experimental results show that Functionally-Predefined Kernel can significantly reduce the number of learnable parameters in CNNs without hurting the recognition accuracy. Furthermore, we observed that many combinations of a multiplier and a multiplicand are repeatedly executed in the inference process and that computation reuse has a high affinity for CNN with Functionally-Predefined Kernel.

Our future work focuses on the design of more versatile Functionally-Predefined Kernels. Additionally, we will develop and evaluate an architecture of computation reuse combined with Functionally-Predefined Kernel.

Acknowledgment

This work is partially supported by JSPS KAKENHI Grant Numbers JP17H01711, JP17H01764, and JP17K19971. This work is also supported by "Joint Usage/Research Center for Interdisciplinary Large-scale Information Infrastructures" and "High Performance Computing Infrastructure" in Japan (Project ID: jh190039-ISH).

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, 2012, pp. 1097–1105.

[2] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," arXiv:1409.1556 [cs.CV], Sep. 2014.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016.

[4] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *Int'l Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[5] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of Deep Convolutional Neural Networks for Fast and Low Power Mobile Applications," arXiv:1511.06530 [cs.CV], Nov. 2015.

[6] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-NET: Imagenet Classification Using Binary Convolutional Neural Networks," in *Proc. European Conf. on Computer Vision*. Springer, 2016, pp. 525–542.

[7] D. Miyashita, E. H. Lee, and B. Murmann, "Convolutional Neural Networks using Logarithmic Data Representation," arXiv:1603.01025 [cs.NE], Mar. 2016. [Online]. Available: http://arxiv.org/abs/1603.01025

[8] T. Chen *et al.*, "DianNao: a Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning," in *Proc. 19th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS'14)*, 2014, pp. 269–284.

[9] S. Shindo, Y. Matsui, H. Yamaki, T. Tsumura, and S. Miwa, "An Acceleration for CNN by Approximate Computation based on Kernel Clustering (in Japanese)," in *IPSJ Technical Report*, vol. 2018-ARC-230, no. 31, Mar. 2018, pp. 1–6.

[10] F. Juefei-Xu, V. N. Boddeti, and M. Savvides, "Local Binary Convolutional Neural Networks," arXiv:1608.06049 [cs.LG], Aug. 2016. [Online]. Available: http://arxiv.org/abs/1608.06049

[11] A. Mahendran and A. Vedaldi, "Understanding Deep Image Representations by Inverting Them," in *Proc. Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015, pp. 5188–5196.

[12] S. Tokui, K. Oono, S. Hido, and J. Clayton, "Chainer: A Next-Generation Open Source Framework for Deep Learning," in *Proc. Workshop on Machine Learning Systems (LearningSys) in 29th Annual Conf. on Neural Information Processing Systems (NIPS)*, 2015.

[13] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. of the IEEE*, vol. 86, pp. 2278–2324, Nov. 1998.

[14] A. Krizhevsky and G. Hinton, "Learning Multiple Layers of Features from Tiny Images," Univ. of Toronto, Tech. Rep., 2009.

[15] N. Otsu and T. Kurita, "A New Scheme for Practical Flexible and Intelligent Vision Systems," in *Proc. IAPR Workshop on Computer Vision*, Oct. 1988, pp. 431–435.

[16] K. Uehara, H. Sakanashi, H. Nosato, M. Murakawa, H. Miyamoto, and R. Nakamura, "Object Detection of Satellite Images using Multi-Channel Higher-Order Local Autocorrelation," in *2017 IEEE Int'l Conf. on Systems, Man, and Cybernetics (SMC)*. IEEE, 2017, pp. 1339–1344.

[17] I. Bulugu, Z. Ye, and J. Banzi, "Higher-Order Local Autocorrelation Feature Extraction Methodology for Hand Gestures Recognition," in *2017 2nd Int'l Conf. on Multimedia and Image Processing (ICMIP)*. IEEE, 2017, pp. 83–87.

[18] X. Jiao, V. Akhlaghi, Y. Jiang, and R. K. Gupta, "Energy-efficient neural networks using approximate computation reuse," in *Proc. 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 1223–1228.