FAST-ABC: A FAST ARCHITECTURE FOR BOTTLENECK-LIKE BASED CONVOLUTIONAL NEURAL NETWORKS

Xiaoru Xie, Fangxuan Sun, Jun Lin and Zhongfeng Wang

School of Electronic Science and Engineering, Nanjing University, P.R. China
{xrxie, fxsun}@smail.nju.edu.cn, {jlin, zfwang}@nju.edu.cn

ABSTRACT

Convolutional neural networks (CNNs) have achieved remarkable performance in many machine learning applications at the cost of high computational complexity. Hence, many efforts have been made to construct light weight networks (LWNs). It is observed that a bottleneck-like operation (BLO) is adopted in many LWNs such as ResNet, MobileNetV2, and ShuffleNet. In this work, an efficient accelerator for BLO-based networks called fast architecture for bottleneck-like based convolutional neural networks (fast-ABC) is proposed. To reduce the computational complexity, Winograd algorithm is employed. Furthermore, the presented BLO based processing elements can be reconfigured to support several convolution kernel sizes. Aiming at reducing memory access, a partial layer fusion strategy is also introduced in this work. Fast-ABC is implemented on FPGA Xilinx Virtex-7 XC7VX485T. Processing speeds of 153 fps and 294 fps can be achieved for Res-18 and MobileNetV2, respectively.

Index Terms— Convolutional neural networks (CNNs), Bottleneck-like operator, FPGA, Winograd

1. INTRODUCTION

Convolutional neural networks (CNNs) have become the most promising and popular machine learning techniques in recent years. Since AlexNet [1] was proposed, more complicated networks were presented in order to achieve better performance. However, the high computational complexity prevents the deployment of large scale CNNs on resource-constrained equipments. To address this problem, many efforts have been made to develop light weight networks (LWNs) such as ResNet [2], MobileNet [3, 4], and ShuffleNet [5].

LWNs usually represent networks with 2 main characteristics: 1) Much less computational and storage complexity. 2) Equivalent or even better accuracy than conventional networks. Table.1 compares existing representative CNNs from these two perspectives. Compared with AlexNet and VGG-16 [6], ResNet and MobileNet series achieve better performances with much less parameters.

Table 1: Comparisons of LWNs and conventional CNNs

	LWNs		Conventional CNNs	
	ResNet-50	**MobileNetV2**	AlexNet	VGG-16
Top1	**77.15%**	**74.7%**	57.2%	71.5%
Params	**20.7M**	**6.9M**	60M	138M
Flops	**3.8G**	**0.59G**	0.72G	15.3G
Depth	50	54	8	16

Aiming at finding how these networks be more efficient on model size and speed, we analysis structures of the most famous LWNs. Table. 2 shows the details of several typical LWNs. There are mainly 3 structures in LWNs: 1) Res-Unit. Two 3×3 convolutional layers are cascaded; 2) Bottleneck in ResNets (BR). Two 1×1 convolution layers and one 3×3 convolutional layer are employed.; 3) Bottleneck in MobileNetV2 and ShuffleNet. The 3×3 convolutional layer in BR is substituted by a 3×3 depth-wise convolutional layer. We can find that all these structures are cascaded by two or three convolutional layers. Kernel sizes of 3×3 and 1×1 are employed in these layers. These operations occupy most of the computation in LWNs ($> 89\%$). Moreover, it can be observed that LWNs are usually much deeper than conventional CNNs.

According to our analysis, bottleneck-like operation (BLO) is defined in this paper to represent an assembly of the three operators mentioned above. Consider that almost all LWNs can be constructed using the cascades of BLO, an efficient architecture for BLO is very necessary for effective inference.

Table 2: Structures of LWNs.

	ResNet-18	ResNet-50	MobileNetV2	ShuffleNet
Operators	Res-Unit	Bottleneck	Bottleneck	Bottleneck
Details[1]	3×3 conv 3×3 conv	1×1 conv 3×3 conv 1×1 conv	1×1 conv 3×3 dwise[2] 1×1 conv	1×1 conv 3×3 dwise 1×1 conv
Shortcut	Yes	Yes	Yes	Yes
OP Ratio[3]	93.4%	97.0%	89.3%	93.5%

[1] **Details** denotes details of a single operator.
[2] **dwise** here denotes depth-wise convolution.
[3] **OP Ratio** denotes ratio of the operators occupy in the whole network.

Previous accelerators for CNNs mainly focus on implementing conventional CNNs while ignoring the optimization for BLO which is the most important part in LWNs. Designing an architecture for LWNs is non-trivial but faces several challenges: 1) Achieving both efficiency and reconfigurability. Although BLOs of various LWNs are similar, there are still some differences among them; 2) Massive memory access. Consider that most accelerators will sent all data to DRAMs after processing a layer, amount of DRAM access will be large for very deep LWNs.

To address these problems, we present an efficient architecture for BLO based LWNs called fast architecture for bottleneck-like based convolutional neural networks (fast-ABC) in this work. Fast-ABC is implemented on FPGA Xilinx Virtex-7 XC7VX485T. According to our experiments, processing speeds of 294 fps, 153 fps, 40 fps for MobileNetV2, Res-18, and Res-50 can be obtained. The main contributions are listed as follows:

1) Based on our analysis of most current LWNs, a bottleneck-like operation is first defined in this work. An optimized processing element which considers both efficiency and reconfigurability for BLO is proposed. Winograd algorithm [7, 8, 9] is well combined with the proposed inference architecture. A significant reduction in algorithmic complexity can thusly be achieved. The presented architecture can be reconfigured to support almost all BLOs used by LWNs.

2) Considering the massive memory access of deep LWNs, a partial fusion strategy which can compute several layers together to reduce data movements is introduced in this work. In typical fusion strategy, manual design of computation flow and memory are needed for every layer in a given network. To avoid exhaustive search of optimal design for different networks, a partial fusion method is presented to achieve a balance between efficiency and reconfigurability.

2. RELATED WORKS

Computationally efficient CNN accelerators focus on reducing the algorithmic strength by employing various fast convolution methods, such as Winograd and fast FIR algorithm (FFA). Based on Winograd method, [10, 11] propose fast convolution architectures to enable efficient computation of some specific convolutions (3×3 and 5×5). In [12], FFA is employed for faster processing of 3×3 convolution. However, extra hardware resources are needed when performing BLOs with various structures. Fast-ABC addresses this problem by proposing a reconfigurable processing element based on Winograd algorithm.

Layer fusion architecture [13, 10] reconstruct the computation of CNNs and compute multiple convolutional layers together. Intermediate data movement between on-chip and off-chip memory can be reduced by employing fusion architecture. However, manual design of computation flow and memory access scheme are needed for different network. Hence, this method only works for relatively small networks with few layers. In this work, we analyze the architecture of LWNs and introduce a partial fusion method. Most current LWNs can be supported by our partial fusion method without huge manually interactions.

Accelerators for ResNets [14, 15, 16] present implementations of ResNets on FPGA. Our fast-ABC aims at supporting almost all LHWs based the optimization of their basic operations. Moreover, efficient Winograd is also used to further reduce the computational complexity.

3. ARCHITECTURE FOR LWNS

In this section, the description of fast-ABC will be given. According to our analysis of structures of LWNs and the definition of BLOs in Section. 1, a fast reconfigurable processing element is proposed for BORs. Winograd algorithm will be employed to further reduce the algorithmic strength. A brief introduction of Winograd algorithm will be given at the beginning of the section. Moreover, to balance reconfigurability and efficiency, a dedicated computational flow using partial fusion method is presented. The overview of the whole architecture will be given at last.

3.1. Winograd Algorithm

Winograd algorithm can implement convolution with less multiplications, especially for small kernels. Assume that an r-tap FIR filter which computes m outputs is denoted as $F(m, r)$. A 2D algorithm $F(m \times m, r \times r)$ can be represented as follows:

$$Y = A^T \left[[GgG^T] \odot [B^T dB] \right] A, \tag{1}$$

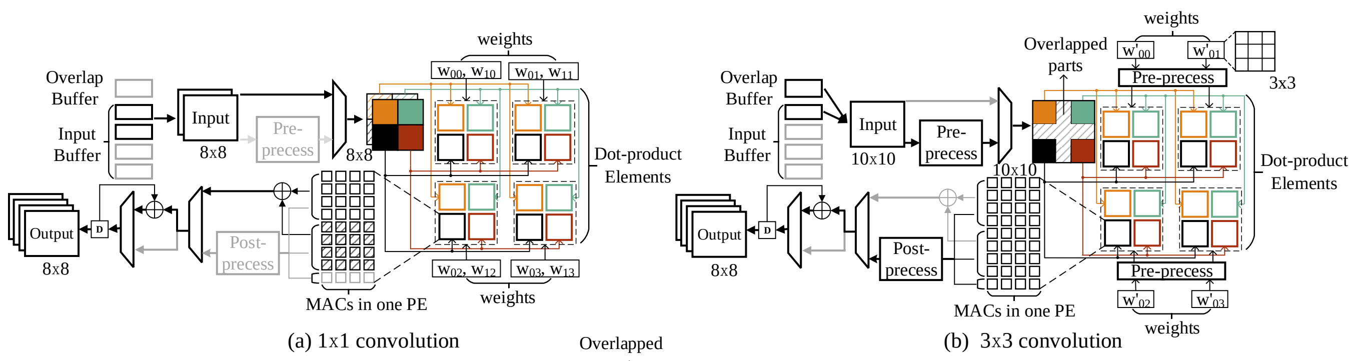
where g is an $r \times r$ filter and d is an $(m + r - 1) \times (m + r - 1)$ image tile. G and B are constant matrices which **pre-process** filter and image tile, respectively. **Post-processing** is implemented by constant matrices A. The details of 2D Winograd algorithm for convolutions can be referred in [17].

In our work, $F(4 \times 4, 3 \times 3)$ is employed as the basis of convolution. According to [17], the number of multiplications can be reduced from 144 to 36.

3.2. Fast Reconfigurable Processing Element

To flexibly support different convolutions used in BLOs, Winograd algorithm is divided into 3 parts: **pre-process, dot-product, and post-process** (details can be referred in Section. 3.1). Pre-processor and post-processor are multiplier-free. Multipliers are only used in dot-product blocks.

Based on our division of Winograd algorithm, a fast reconfigurable processing element (FRPE) for BLO is presented. Fig. 1 illustrates how FPRE computes different convolutional layers which are introduced in Section. 1. The introductions of each part of FRPE are listed as follows: 1)



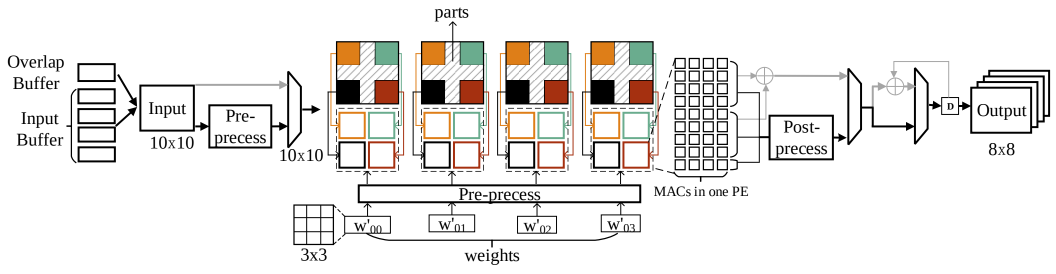


Fig. 1: Details of FRPE. Three computation modules are illustrated. (a), (b), and (c) show how 1×1 convolution, 3×3 convolution, and 3×3 depth-wise convolution are computed, respectively.

Table 3: Computation Procedures for Various Convolutions.

	1×1 conv	3×3 conv	3×3 dwise
Inputs size	$8 \times 8 \times 2$	$10 \times 10 \times 1$	$10 \times 10 \times 4$
Pre-process	No	Yes	Yes
Dot-product (per DPE)	2 CHs computed 16 Muls each CH	1 CH computed 36 Muls each CH	1 CH computed 36 Muls each CH
Post-process	No	Yes	Yes
Accumulate[1]	Yes	Yes	No
Outputs	$8 \times 8 \times 4$ Sum of 2 CHs[2]	$8 \times 8 \times 4$ Sum of 1 CH	$8 \times 8 \times 4$ Sum of 1 CH

[1] Accumulate here denotes whether accumulations are needed across channels.
[2] CHs denote channels.

Overlap buffer is an on-chip memory which are used to store overlapped part between two adjacent input tiles. It is worth noting that overlapped parts do not exist in 1×1 convolution, the overlap buffer is thusly disabled when computing 1×1 convolution. 2) **Input buffer** stores input activations. In our work, data of different sizes are needed in various configuration, 4 memory banks are employed. 3) **Dot-product Element (DPE)** has 36 multiplications ($F(4 \times 4, 3 \times 3)$). There are 16 DPEs in FRPE. **Weight**$_{xy}$ denotes a convolution kernel of x-th input channel and y-th output channel. Size of kernels is 1×1 in Fig. 1(a) while is 3×3 in Fig. 1(b) and Fig. 1(c).

The computation procedures of these 3 types of convolutions are listed in Table. 3. Among them, pre-processor and post-processor are enabled for 3×3 convolutions (including depth-wise convolution). To enhance the resource utilization when computing 1×1 convolution, two tiles are computed together in a single DPE. The utilization of multipliers for 1×1 convolution 88.9% (32 multipliers in a DPE are used.). For all three convolutions, the sizes of outputs are the same ($8 \times 8 \times 4$).

Algorithm 1: Partial fusion computational flow.

```
Input  : $I, W$
Output: $O$
1 $I' = I$
2 for $i = 1$ to $N_{BRO}$ do
3     for $j = 1$ to $N_{tile}$ do
4         for $k = 1$ to $N_{layer}$ do
5             for $m = 1$ to $N_{out_ch}$ do
6                 for $n = 1$ to $N_{in_ch}$ do
7                     $I'(k+1, m, j)$+=conv($I'(k, n, j)$, $W(i, k, m, n)$)
8 $O = I'(N_{layer}, :, :)$
```

3.3. Partial Fusion Computational Flow

As mentioned in Section. 1, there are two main problems existing in the designing architectures for LWNs. To address these problems, a partial layer fusion method is proposed. In this work, we compute all layers in a BLO together. Details of the computation of BLO is shown in Algo. 1, where I and W

denote input activation and weight, respectively. O is used to denote final output. N_{BRO}, N_{tile}, N_{layer}, N_{out_ch}, and N_{in_ch} denote the number of BROs, tiles in a layer, layers in one BRO, input channels of current layer, and output channels of current layer, respectively.

3.4. Overview

The overview of our architecture for BLO based LWNs is shown in Fig. 2. Blocks which are not introduced above will be discussed as follows: **Buffer1-Buffer3** are used to store activations. Among them, Buffer1 stores the input data of first layer in LWNs. Buffer2 is used to store the data of last layer in LWNs. Since a BLO consists of two or three layers, Buffer3 is employed to store the intermediate data. Due to the the existing of shortcut in BROs, the data in Buffer1 will be kept until being added to the outputs of the last layer (stored in Buffer2) in BROs. **Overlap Buffer** is employed to store the overlaps between tiles. **WReg** stores weights which are fetched from DRAM. **PReg** is used to stored the partial sums during the computation of convolution. **ReLU Unit** performs the ReLU function after convolution.

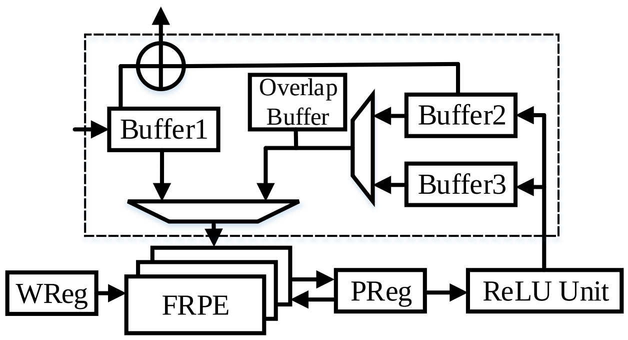


Fig. 2: Overview Architecture.

4. EXPERIMENTS AND COMPARISONS

The proposed fast-ABC is implemented on Xilinx Virtex7 VC707 FPGA. Performance of fast-ABC and comparisons with previous works will be shown in this section.

FRPE: In previous works which employ Winograd algorithm [11, 10], processing elements are designed for 3×3 convolution. However, extra resources are needed for BLOs with various structures if directly applying LWNs on those previous Winograd processing elements. In our work, the proposed FRPE can be reconfigured to support various convolutions effectively. According to Table. 1, three typical BRO structures (Res-Unit, Bottleneck dwise, and Bottleneck) are employed by Res-18, MobileNetV2, and Res-50, respectively. Hence, these three networks are selected for experiments.

Hardware performances of different networks are shown in Table. 4. Consider that BLOs occupy almost all the computation of LWNs, our experiments are only conducted on the BLO parts. As shown in Table. 4, throughputs of 262.8 GOP/s, 85.9 GOP/s, and 159.8 GOP/s can be achieved for Res-18, MobileNetV2, and Res-50. Among them, fast-ABC achieves a better throughput on Res-18 which is employs Res-Unit as the essential operation. For MobileNetV2, fast-ABC can obtain a 3.4ms latency which is the fastest among the LWNs listed in Table. 4.

Table. 4 also compares fast-ABC with previous works on ResNet. Our work can achieve a better throughput compared with previous works. Moreover, fast-ABC is implemented with much less multipliers. Our works outperforms previous works in computational efficiency by 34%-337%.

Table 4: Comparisons with previous works.

	[15]	[16]	ours
Model	Res-50	Res-152	-
FPGA	Stratix V GXA7	Stratix V GSMD5	Virtex-7 XC7Z045T
Freq (MHz)	150	150	178
Precision	16bit fixed	16bit fixed	16bit fixed
Mul	1,176	1,036	576
DSP	256	1,036	576
LUT/ALM[1]	176K	45.7K	195K
BRAM[2]	1,950	959	222
Latency (ms)	31.82	-	**6.5** (Res-18)
			3.4 (MoblieNetV2)
			24.5 (Res-50)
Throughput (GOP/s)	121.6	113.2	**259.9** (Res-18)
			84.9 (MoblieNetV2)
			158.0 (Res-50)
Efficiency (GOP/s/Mul)	0.103	0.109	**0.451** (Res-18)
			0.147 (MoblieNetV2)
			0.274 (Res-50)
Normalized Efficiency	1	1.06	**4.37** (Res-18)
			1.42 (MoblieNetV2)
			2.45 (Res-50)

[1] LUT and ALM are used in Xilinx and Altera platform, respectively.
[2] Sizes of block RAM unit in Xilinx and Altera platform are 20Kb and 36Kb, respectively.

5. CONCLUSION

In this work, a bottleneck-like operation which is mainly adopted by light weight networks is first defined. An efficient accelerator for BLO-based networks called fast-ABC is proposed. A fast reconfigurable processing element based on Winograd algorithm is presented. BLOs with different structures can be supported by FRPE. Moreover, a partial fusion computational flow which balances the reconfigurability and memory access. Fast-ABC is implemented on FPGA Xilinx Virtex-7 XC7VX485T. Our work achieves better throughputs than previous works on ResNets. Moreover, compared with previous works, improvements of 34%-337% on computational efficiency are achieved.

6. REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," pp. 770–778, 2016.

[3] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[4] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.

[5] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: an extremely efficient convolutional neural network for mobile devices," in *Computer Vision and Pattern Recognition*, 2017.

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[7] S. Cook, "On the minimum computation time for multiplication," *Doctoral diss., Harvard U., Cambridge, Mass*, vol. 1, 1966.

[8] A. L. Toom, "The complexity of a scheme of functional elements realizing the multiplication of integers," in *Soviet Mathematics Doklady*, 1963, vol. 3, pp. 714–716.

[9] S. Winograd, *Arithmetic complexity of computations*, vol. 33, Siam, 1980.

[10] Q. Xiao, Y. Liang, L. Lu, S. Yan, and Y. Tai, "Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on fpgas," in *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, 2017, p. 62.

[11] L .Lu, Y. Liang, Q. Xiao, and S. Yan, "Evaluating fast algorithms for convolutional neural networks on fpgas," in *Field-Programmable Custom Computing Machines (FCCM), 2017 IEEE 25th Annual International Symposium on*. IEEE, 2017, pp. 101–108.

[12] J. Wang, J. Lin, and Z. Wang, "Efficient hardware architectures for deep convolutional neural network," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 6, pp. 1941–1953, 2018.

[13] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer cnn accelerators," in *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Press, 2016, p. 22.

[14] Y. Ma, M. Kim, Y. Cao, S. Vrudhula, and J. Seo, "End-to-end scalable fpga accelerator for deep residual networks," in *Circuits and Systems (ISCAS), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 1–4.

[15] Y. Ma, Y. Cao, S. Vrudhula, and J. Seo, "Optimizing the convolution operation to accelerate deep neural networks on fpga," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, , no. 99, pp. 1–14, 2018.

[16] Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang, and J. Cong, "Fp-dnn: An automated framework for mapping deep neural networks onto fpgas with rtl-hls hybrid templates," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2017, pp. 152–159.

[17] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4013–4021.