EADNET: EFFICIENT ARCHITECTURE FOR DECOMPOSED CONVOLUTIONAL NEURAL NETWORKS

Fangxuan Sun, Jun Lin and Zhongfeng Wang

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

ABSTRACT

Convolutional neural networks (CNNs) are widely used in various intelligent tasks. However, the huge computational complexity of CNNs makes it hard to be implemented in many real-time embedded devices. Various methods have been employed to reduce the model size of CNNs, where the Canonical Polyadic Decomposition (CPD) has shown its capability to reduce both the computational complexity and the storage requirement with negligible accuracy loss. In this paper, an efficient configurable hardware architecture called EadNet is proposed for CPD-CNNs. In detail, to minimize the on-chip memory access, different data reuse patterns are first analyzed. Based on the chosen optimal reuse scheme, a much improved computation flow is also developed for efficiently caching activations. The EadNet is implemented with a TSMC 90nm CMOS technology. The implementation results indicate that EadNet achieves considerable improvements on computation efficiency compared to the state-of-the-art CNN accelerator architectures.

Index Terms— Deep Learning, Convolutional Neural Networks (CNNs), CP Decomposition, VLSI

1. INTRODUCTION

Convolutional neural networks (CNNs) have achieved remarkable performance in many fields, such as computer vision and speech recognition [1–3]. The improvement of model capability of CNNs leads to more convolutional layers, which incurs higher computational complexity and more memory requirement. Dedicated hardware architectures for CNNs are necessary for power stringent embedded applications. Due to limited hardware resource and memory bandwidth, embedded hardware accelerators suffer from the huge computational complexity of modern deep CNNs.

For the purpose of speeding up CNNs, some decomposition methods have been proposed for CNNs [4–10]. Among them, Canonical Polyadic decomposition (CPD) was presented in [5, 7]. The weight size and computational complexity can be reduced by 6.98x and 3.53x, respectively, with only 1.42% accuracy drop, which is known as the state-of-the-art performance among decomposed CNNs. Hence, deploying CPD-CNNs on processors or specific hardware accelerators is necessary.

In CPD-CNNs, the decomposed comvolutional layer will be split into several low rank sub layers with less weight size. On the other hand, the total number of activations are inevitably increased due to the generated sub layers. Therefore, accelerators designed with CPD should focus on optimizing the activation access from memory. However, many current accelerators pay much attention on minimizing the weight access [11–21]. An architecture with reconfigurable computation patterns was presented in [11], the memory access was analyzed and a hybrid reuse pattern was proposed. Chen *et al.* [12] proposed a row stationary data flow which was used for CNNs on a spatial architecture. A data flow which fused the processing of multiple CNN layers with minimal DRAM access was proposed in [13]. Directly applying current architectures to CP-decomposed CNNs will incur unnecessary on-chip memory and extra data access.

In this paper, the structural characteristics of decomposed CNNs are explored and different data reuse schemes are analyzed to develop the best data reuse scheme for CPD-CNNs. Based on the chosen reuse scheme, a much improved computation flow which can efficiently cache activations and reduce the size of on-chip memory is proposed. Combined with the CPD, the presented computation flow also eases the requirement of DRAM bandwidth. Finally, based on selected reuse scheme and computation flow, an efficient hardware architecture called EadNet is proposed to support various convolution kernel sizes. To the best of our knowledge, EadNet is the first dedicated hardware architecture for CPD-CNNs. The implementation results demonstrate that EadNet has 4.9 to 9 times better computation efficiency compared to the state-of-the-art CNNs architectures.

2. BACKGROUND

For a typical convolutional layer, let I and O denote the input and output feature maps with C_{in} and C_{out} channels, respectively. Let W and H denote the width and height of each output feature map, respectively. For $c_{out} = 0, 1, \cdots, C_{out}$, $w' = 0, 1, \cdots, W-1$, $h' = 0, 1, \cdots, H-1$, an output acti-

vation

$$O_{c_{out},w',h'} = \sum_{i=1}^{d}\sum_{j=1}^{d}\sum_{c_{in}=1}^{C_{in}} K_{c_{out},c_{in},j,i} I_{c_{in},w_j,h_i} \quad , \qquad (1)$$

where K is a 4D kernel tensor of size $d \times d \times C_i \times C_o$. Here, each convolution kernel is a $d \times d$ matrix.

By employing the R-rank CPD, K can be decomposed to three low rank tensors: K^{hw}, K^s, and K^t. The sizes of K^{hw}, K^s, and K^t are $d \times d \times R$, $R \times c_{in}$ and $c_{out} \times R$, respectively. Hence, a convolutional layer can be decomposed into three inter concatenated convolutional layers (ICLs) shown as follows:

$$\begin{aligned} I^s_{r,w,h} &= \sum_{c_{in}=1}^{C_i} K^s_{r,c_{in}} I_{c_{in},w,h} \\ I^{shw}_{r,w',h'} &= \sum_{i=1}^{d}\sum_{j=1}^{d} K^{hw}_{r,i,j} I^s_{r,w_j,h_i} \quad , \\ O_{(c_{out},w',h')} &= \sum_{r=1}^{R} K^t_{c_{out},r} I^{shw}_{r,w',h'} \end{aligned} \qquad (2)$$

where I^s and I^{shw} are the outputs of the first two ICLs. K^s and K^t are both 1×1 convolutional kernels. For the calculation of I_{shw}, it should be noted that the i-th output feature map will be connected to only the i-th input feature map [7]. Based on the CPD method, the compression rate of the AlexNet is shown in Tab. 1.

Table 1. Performance of CPD-CNNs [7].

	Top-5 (%)	Weights (M)	Comp Cost (M)
AlexNet	79.95	61.0	724
CPD-CNNs	78.53	8.7	205

3. EFFICIENT ARCHITECTURE FOR CPD-CNNS

As mentioned in [12], the energy consumption of a CNN processor is mainly attributed to the large amount of computations and memory accesses. CPD can lead to significant reduction in both the computational complexity and model size of a CNN. However, as shown in Eq. (2), CPD significantly increases the number of generated activations during the decomposition procedure, resulting in potential more data movements between memory and computation units when hardware implementations are considered. In this paper, for CPD-CNNs, an efficient computation flow is proposed to enable through reuse of output activations of ICLs.

3.1. Data Reuse

Data reuse [11], which includes input reuse, output reuse, weight reuse, is a common approach to eliminate redundant memory access for various CNN processors or accelerators. In this work, by analyzing the model structure of CPD-CNNs, it is found that reusing output activations can minimize both SRAM and DRAM accesses when the hardware implementation of a CPD-CNN is considered. In more details, different reuse schemes are compared as follows.

Considering limited on-chip hardware resource, feature maps are tiled during processing to reduce the on-chip storage and bandwidth requirement. Input feature maps with a size of $H \times W \times N_{in}$ are tiled as $T_H \times T_W \times T_{in}$. For output feature maps, the size and the tiled size are $R \times C \times N_{out}$ and $T_R \times T_C \times T_{out}$, respectively. The comparisons between different reuse schemes are shown below[1]:

Weight Reuse (WR): The fetched weights are fully utilized before fetching next batch of kernel weights. Partial sums need to be read and written repetitively between computation units and on-chip buffers. The times of data accessed

$$\begin{aligned} MT_w^{WR} &= \lceil\frac{R}{T_R}\rceil\lceil\frac{C}{T_C}\rceil, \quad MT_i^{WR} = \lceil\frac{N_{out}}{T_{out}}\rceil, \\ MT_o^{WR} &= 2(\lceil\frac{N_{in}}{T_{in}} - 1\rceil), \quad MT_{iDRAM}^{WR} = 1, \end{aligned} \qquad (3)$$

where MT_w^{WR}, MT_i^{WR} and MT_o^{WR} denote the times of SRAM accesses of weights, inputs and outputs, respectively. MT_{iDRAM}^{WR} denotes the DRAM access of input activations.

Output Reuse (OR): T_{out} tiled partial sums are reused and stored in registers of MACs until all input feature maps are fetched and processed. The main difference between output reuse and weight reuse occurs in MT_o^{OR}. The times of output activation accesses of output reuse are $MT_o^{OR} = 1$.

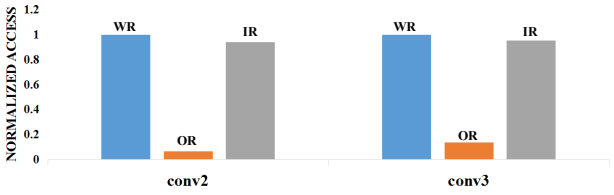

Fig. 1. Comparison of normalized SRAM accesses of different convolutional layers.

Input Reuse (IR): The memory access of input reuse is close to that of output reuse. However, writing partial sums to on-chip buffer would incur extra power consumption.

When the number of activations is much larger than weights and MT_o^{WR} is larger than MT_i^{OR}, output reuse is better than other reuse schemes. For CPD-CNNs, the number of activations is usually larger than that of weights. For example, the total number of activations is 8.3x larger than that of

[1]For simplicity, we assume that $[T_H, T_W]$ equals $[T_R, T_C]$ and $T_{in} \times T_H \times T_W \leqslant B$, where B denotes the on-chip buffer of input activations.

weights in convolutional layers of CPD-AlexNet[2]. Besides, it is worth mentioning that weight reuse scheme was employed in many works to avoid repeatedly access of the same weight from DRAM. For CPD-CNNs, the weights of convolutional layers after decomposition could be stored on-chip if the compression ratio is large enough. The comparison of the numbers of SRAM accesses for CPD-AlexNet under various reuse schemes is shown in Fig. 1. It can be concluded that output reuse scheme is more suitable for CPD-CNNs.

3.2. Computation Flow

During the computation of adjacent ICLs, off-chip DRAM accesses are needed to obtain the activations based on current popular reconfigurable CNN processor architectures, which usually send all output feature maps to DRAM before performing the next convolutional layer. To avoid extra DRAM access and reduce the on-chip memory for caching activations, an efficient computation flow for CPD-CNNs is proposed in this section. In detail, the special structure of CPD-CNNs is analyzed first. For original CNNs, the computation of each output feature map needs all input feature maps, which means N_{in} tiled feature maps are used to process the next layer. However, for CPD-CNNs, during the calculation of I_{shw}, the i-th output feature map will be connected to only the i-th input feature map as shown in Eq. (2). Hence, the obtained I_s can be immediately used to compute I_{shw} rather than being written into an SRAM. The proposed efficient computation flow is illustrated in Fig. 2(a). Assuming the number of processing arrays (PAs) is N_P and each PA takes charge of the computation of an output channel. The proposed computation flow is described as follows.

Step1. Before computing a convolutional layer, N_{in} tiled input feature maps are read to on-chip buffer from DRAM. These N_{in} maps are sent to PAs in serial to get N_P tiled feature maps of I_s. During this step, convolutions with 1×1 kernels are employed.

Step2. According to Eq. (2), the N_P tiled I_s are directly used for the computation of I_{shw} rather than being buffered. N_P tiled maps of I_{shw} will be obtained and sent to on-chip buffers. PAs are used for the calculation of I_s and I_{shw} iteratively until all R tiled feature maps of I_{shw} are calculated. To avoid large on-chip memory and extra DRAM access, only $R \times T_H \times T_W$ activations of I_{shw} are computed and buffered. During this step, convolutions with $d \times d$ kernels are performed.

Step3. After all R tiled maps of I_{shw} are gotten, N_{out} tiled output feature maps will be calculated and sent to off-chip DRAM sequentially. Meanwhile, a new batch of tiled feature maps of I are fetched from DRAM during ***Step3*** to reduce the requirement of bandwidth of DRAM. The proposed computation flow will jump to ***Step1*** after finishing ***Step3***.

[2]The trained model and source code is provided by Marcella Astrid, an author of [7]. ImageNet is used as dataset.

The above described process will be repeated for $\lceil \frac{H}{T_H} \rceil \lceil \frac{W}{T_W} \rceil$ times to finish the calculation of a decomposed convolutional layer. In order to reduce the inference time, the DRAM bandwidth should be large enough so that the new batch of activations can be loaded into the on-chip buffers before the calculation of N_{out} tiled output feature maps is finished.

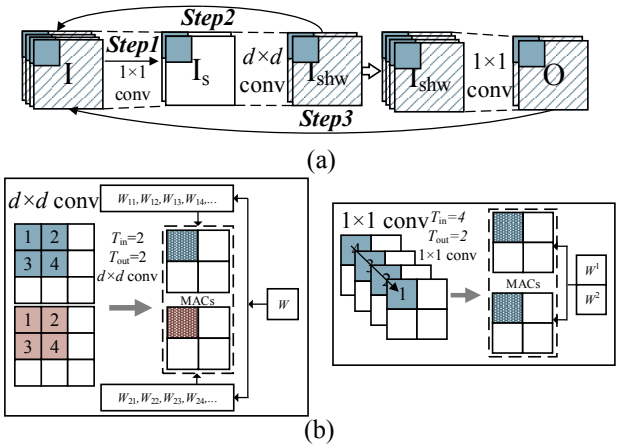

Fig. 2. Data Flow. (a) The computation flow of EadNet. Colored small blocks denote tiled activations. Feature maps with stripes mean that these maps are sent to on-chip buffer after calculation. (b) Computation patterns. The colored inputs are sent to the MAC with the same color in serial.

3.3. Hardware Architecture and Computation Pattern

The efficient hardware architecture for CPD-CNNs, namely EadNet is presented in this section as shown in Fig. 3. The convolution weights can be stored on-chip due to the application of CP decomposition. The Weight Buffer unit is an SRAM which stores all convolution weights. For each convolutional layer, the size of weights is $R(N_{in} + N_{out} + d^2)$. The Activation Buffer unit stores temporary activations. Assuming that B_{in} tiled input feature maps are buffered, the total size of the activation buffer is $B_{in} \times T_H \times T_W$. If $B_{in} \geq N_{in}$, each activation will be read from DRAM only once. Otherwise, each activation will be read $\lceil \frac{N_{out}}{T_{out}} \rceil$ times. Each Tiled Registers (TR) unit stores a $T_H \times T_W$ tiled feature map. In this work, T_{in} TR units are employed.

Each PA unit contains $T_R \times T_C$ Multiply Accumulation Unit (MAC). PA is used for all computations in convolutional layers. These PAs can be configured to support different kernel sizes to adapt various networks. Fig. 2(b) illustrates two computation patterns for $d \times d$ and 1×1 convolution, respectively. A output pixel is assigned to a single MAC. The needed inputs are sent to the MAC in serial by the order shown in Fig. 2(b), where corresponding inputs and MACs

Table 2. Hardware Resources And Performance Evaluated On AlexNet*.

	Layer	Technology (nm)	Area (mm^2)	SRAM (KB)	Frequency (MHz)	MACs Num.	Power (mW)	T_{typ} (TOP/s)	BW (GB/s)	Efficiency (layers/s/MAC)	Normalized Comparison†
[12]	CONV2	65	12.25	-	200	168	288	0.067	-	0.57	1
[11]	CONV2	65	16	-	200	512	-	0.205	-	0.81	1.42
EadNet	CONV2	90	6.11	150	600	256	286	3.049	2.76	3.98	6.98
[12]	CONV3	65	12.25	-	200	168	266	0.067	-	1.01	1
[11]	CONV3	65	16	-	200	512	-	0.205	-	1.13	1.12
EadNet	CONV3	90	9.05	212	700	256	507	3.19	2.62	10.28	9.10

* Decomposition ranks are 154 and 153 for CONV2 and CONV3, respectively. The supply voltage is 1.0V. Precision is set as 16-bit fixed-point.
† This column is normalized Efficiency for better comparison.

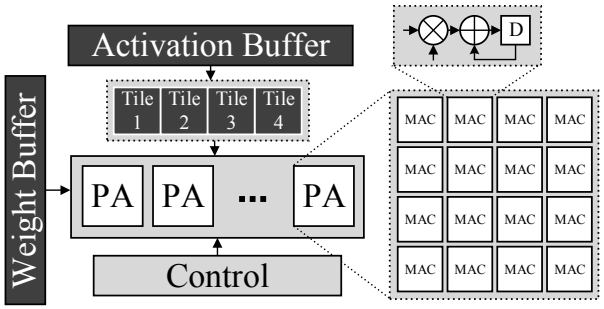

Fig. 3. Overview of architecture. Blocks colored with white denote the computation units, dark blocks are used to represent on-chip buffer.

share the same color. When kernel size is less than the number of MACs in a PA, the outputs can be directly computed. Otherwise, four PAs will be combined together to get a joint PA. The utilization ratio of PAs will not drop by doing so.

4. RESULTS AND COMPARISONS

The hardware complexity and performance of EadNet are evaluated in this section. For the proposed architecture, T_{in}, T_H, and T_W are set to 16, 4 and 4, respectively. The number of PAs is 16, where each PA contains 16 MACs. Weights of different convolutional layers are stored on-chip.

According to Section. 3.2, the requirement of DDR bandwidth can be eased due to CPD and the proposed efficient computation flow. In ***Step 3***, the calculation of O needs at least $\frac{C_{out}R}{T_{in}}$ cycles, which means the requirement of DDR bandwidth is $BW = \frac{2(C_{in}+C_{out})T_HT_WT_{in}f}{C_{out}R}$, where f denotes the frequency. In general, C_{in} is no more than C_{out}, so $BW \leq \frac{4T_HT_WT_{in}f}{R}$.

Since none architectures have been proposed for CPD-CNN before, two representative works [11, 12] are used for comparison as shown in Table 2. Two layers of AlexNet with the highest computational complexity are chosen for evaluation. Activation sparsity is utilized to reduce power consumption in [12]. In this work, the sparsity is not considered. However, activation/weight sparsity and the decomposition can be seen as complement to each other. Combining these two schemes can produce more compact models. T_{peak} denotes the peak throughput and T_{typ} is used to denote the throughput of corresponding convolution without decomposition. T_{peak} and T_{typ} are given by the following equations:

$$T_{peak} = (N_m + N_a) * f, \quad T_{typ} = \frac{T_{peak}}{ratio_{cp}} \quad , \tag{4}$$

where N_m and N_a denote the number of multipliers and adders in all PAs, respectively. The compression ratio of operations is denoted by $ratio_{cp}$. The efficiency of EadNet in Tab. 2 does not take the ReLU and pooling into consideration.

Some works without layer-wise results or evaluated on different dataset are compared as below: Du *et al.* [14] introduced a methodology of configuring the large-sized kernel computation with many small-sized kernels computation, which can be seen as a complement of our computation pattern. Since only LeNet was evaluated in [14] whose computational complexity and model size are relatively small, the performance on modern CNNs are unknown. In addition, no efficient network compression scheme was employed, which leads to more memory requirement and higher computational complexity. In [15], a dual-range MAC was developed for low-power convolution. PCA was employed to compress the kernel data, which resembles ours. However, the computational complexity is not reduced.

5. CONCLUSION

In this paper, carefully selected data reuse scheme and an improved computation flow are developed for CPD-CNNs. Moreover, an efficient configurable hardware architecture called EadNet is proposed for CPD-CNNs. For EadNet, the number of memory access is minimized and various kernel sizes are supported. The proposed architecture is coded with RTL and synthesized under a TSMC 90nm CMOS technology. The implementation results show that the EadNet is many times better than representative CNN architectures in terms of computation efficiency.

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

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[4] Y. 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 preprint arXiv:1511.06530*, 2015.

[5] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," *arXiv preprint arXiv:1412.6553*, 2014.

[6] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," *arXiv preprint arXiv:1405.3866*, 2014.

[7] M. Astrid and S. Lee, "Cp-decomposition with tensor power method for convolutional neural networks compression," in *Big Data and Smart Computing (BigComp), 2017 IEEE International Conference on*. IEEE, 2017, pp. 115–118.

[8] W. Wen, C. Xu, C. Wu, Y. Wang, Y. Chen, and H. Li, "Coordinating filters for faster deep neural networks," *arXiv preprint arXiv:1703.09746*, 2017.

[9] T. Garipov, D. Podoprikhin, A. Novikov, and D. Vetrov, "Ultimate tensorization: compressing convolutional and fc layers alike," *arXiv preprint arXiv:1611.03214*, 2016.

[10] E. Denton, W. Zaremba, J. Bruna, Y. Lecun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," *International Conference on Neural Information Processing Systems*, pp. 1269–1277, 2014.

[11] F. Tu, S. Yin, O. Peng, S. Tang, L. Liu, and S. Wei, "Deep convolutional neural network architecture with reconfigurable computation patterns," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2017.

[12] Y. Chen, K. Tushar, S. E. Joel, and S. Vivienne, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[13] A. Manoj, C. Han, F. Michael, and M. Peter, "Fused-layer cnn accelerators," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016.

[14] L. Du, Y. Du, Y. Li, and M. Chang, "A reconfigurable streaming deep convolutional neural network accelerator for internet of things," *IEEE Transactions on Circuits and Systems I-regular Papers*, pp. 1–11, 2017.

[15] J. Sim, J. Park, M. Kim, D. Bae, Y. Choi, and L. Kim, "14.6 a 1.42tops/w deep convolutional neural network recognition processor for intelligent ioe systems," *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 264–265, 2016.

[16] R. Brandon, W. Paul, A. Robert, R. Saketh, H. Lee, S. Lee, J. Hernndez-Lobato, G. Wei, and B. David, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, vol. 44, no. 3, pp. 267–278, 2016.

[17] L. Song, Y. Wang, Y. Han, X. Zhao, B. Liu, and X. Li, "C-brain: A deep learning accelerator that tames the diversity of cnns through adaptive data-level parallelization," *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, p. 123, 2016.

[18] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, and S. Song, "Going deeper with embedded fpga platform for convolutional neural network," *Acm/sigda International Symposium on Field-Programmable Gate Arrays*, pp. 26–35, 2016.

[19] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh, "From high-level deep neural models to fpgas," *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–12, 2016.

[20] M. Motamedi, P. Gysel, V. Akella, and S. Ghiasi, "Design space exploration of fpga-based deep convolutional neural networks," *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 575–580, 2016.

[21] Y. Shen, M. Ferdman, and P. Milder, "Overcoming resource underutilization in spatial cnn accelerators," *International Conference on Field Programmable Logic and Applications*, pp. 1–4, 2016.