Accelerator Design for Convolutional Neural Network with Vertical Data Streaming

Conference Paper · October 2018

DOI: 10.1109/APCCAS.2018.8605662

Zheng Wang
Shenzhen Institute of Advanced Technology (SIAT), Shenzhen, China

Accelerator Design for Convolutional Neural Network with Vertical Data Streaming

Shanliao Li, Ouyang Ning
School of Information and Communication
Guilin University of Electronic Technology, Guilin, China
562784494@qq.com

Zheng Wang
Shenzhen Institutes of Advanced Technology
Chinese Academy of Science, Shenzhen, China
zheng.wang@siat.ac.cn

***Abstract*—Convolutional neural network (CNN) plays a crucial role in object classification, detection, semantic segmentation, and decision-making. Due to the enormous amount of computing operations and network parameters, CNN demands dedicated hardware accelerators to address the bottlenecks in performance and power consumption. State-of-the-art computing engines for CNN carefully delivers data to processing elements, which requires complicated control logic. In this work, we reduce such control complexity by utilizing the streaming architecture for both data and parameters. By introducing the in-depth addressing mode, the proposed architecture achieves high utilization of processing elements through the sharing of input feature maps. Detailed performance profiling verifies that our design, which is synthesized at only 100 MHz, executes 0.67 frames/sec for all convolution layers of the VGG-16 network with a low power estimate of 20.035 mW.**

***Index Terms*—convolutional neural network, data-streaming architecture, in-depth addressing.**

I. Introduction

In the past years, rapid progress in neural networks has been witnessed in various application domains. Especially in computer vision, the classification and detection based on Convolutional Neural Network (CNN) and its variants have achieved significant success. CNN was proposed in [1] to extract features through a concatenated layers of convolution, pooling and full connection operations. While heavy research bodies have been conducted to improve the accuracy of classification which leads to large CNN network structures such as VGG-16 [2] and VGG-19 [2], another important trend is to extend the basic CNN functionality to support region detection [3] [4] and semantic segmentation [5] [6]. Furthermore, CNN combined with other types of networks such as LSTM and reinforcement learning have resulted in end-to-end networks for context description [7] and human-level gaming agent [8]. The work in [9] shows that 90% computing operations of the state-of-the-art neural networks are based on convolution, which demonstrated the importance of CNN.

Conventionally, CNN-based deep learning algorithms are deployed on CPU-GPU platforms. Recently, CPU-FPGA based heterogenous programming paradigm has also appeared for supporting higher performance. However, both solutions suffer from extensive power consumption caused by significant amount of data movement between heterogenous computing elements. To address this, dedicated hardware accelerators have been proposed to accelerate inference mode of neural networks such as Eyeriss [10], Google TPU-I [11] and Cambricon [12]. While TPU-I adopts systolic array multipliers to compute convolution layer, Eyeriss introduces hardware resource sharing techniques such as reuse of input feature maps and convolution filters, which provides improved resource utilization for convolution layer.

Among various design options for neural networks, data-streaming architecture has been adopted in [13] [14], where data and/or parameters are streamed in and out to share computing resources. One of the key limitations to improve the computing efficiency for convolution in data-stream architecture is the sparsely connected network. Each node in the output feature map is only connected with several nodes in each input feature map, which demands continuously update of input data address for different output node. In contrast,a fully connected layer has a dense connection where all input data are identically required for any output data, therefore the same flow of data can be streamed into the CNN engine and shared by processing elements. In this work, we investigate the properties of convolution layer cross multiple channels and propose the vertical data-streaming architecture by introducing the in-depth addressing mode as opposed to the conventional horizontal addressing mode. By applying such technique on our in-house developed data-streaming accelerator, we demonstrate that in-depth addressing significantly improves the resource usage while simplifies design complexity of the data address generator.

The paper is organized as following. Section II provides the background for the computing of convolution layer and current techniques for optimizing convolution through resource sharing. Section III presents the proposed in-depth addressing mode and its realization on the data-streaming processor. Section IV presents the experiment results on performance and physical properties. Section V concludes the work while proposes future directions.

II. Convolution layer and data sharing

a) Convolution layer: Convolution layer plays the key role in feature extraction for CNN network. A typical convolution layer processes multiple channels of input feature maps (ifmaps) and generates multiple channels of output feature maps (ofmaps). As shown in Figure 1, the convolution layer has N channels of ifmaps with width W and height H, a set of N filters with the individual dimension of $K \times K$ slides through N ifmaps to generate one single ofmap with width C and height R. The stride S defines the step-size of filter movement during sliding of a filter, which is less or equal to K. For each of the M ofmaps there exists a different set of filters. Each node in ofmap has a

This work was supported by NSFC with Grant No. 61702493, Shenzhen S&T Funding with Grant No. KQJSCX20170731163915914, SIAT Innovation Program for Excellent Young Researchers (2017001). Zheng Wang is corresponding author (zheng.wang@siat.ac.cn)

region of projection (RoP) on the ifmaps. Adjacent nodes in one channel of ofmap ,as well as nodes at the same position across different ofmaps have overlapped RoP.

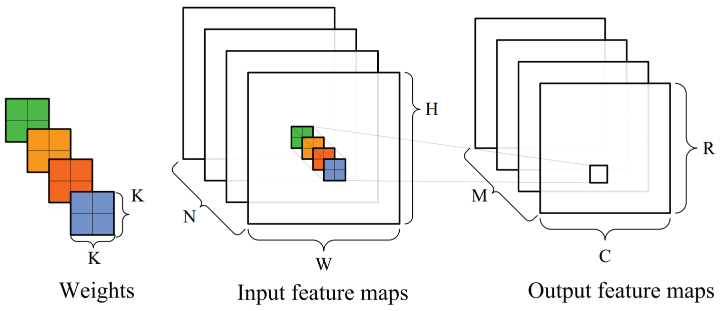

Fig. 1: Graphical representation of convolution layer

b) Data sharing techniques during convolution: State-of-the-art sharing techniques for stream processing of convolution layer are present in for both data and parameters. First and foremost, filters are shared during sliding window for the same pair of ifmap-ofmap. Such filter sharing can be extended to the simultaneously processing of multiple input data samples. Furthermore, the same ifmap needs to be processed by all ofmaps.

III. Vertical data-stream processing

Ideally, the ifmaps should be streamed-in ***only once*** during the design of data-streaming architecture, which is followed by the streaming-out phase of ofmaps. Repeated streaming-in of same data segment would lead to decreased resource utilization and waste in memory bandwidth. However, due to the limitation in the number of processing elements (PEs), only limited nodes in the ofmaps are computed simultaneously by sharing the same input ifmap stream. Such batch of ofmap nodes needs to be streamed-out before the next batch of ofmap nodes can be processed. Consequently, multiple batches of output ofmap streams require repeat streaming-in of ifmaps. Due to the fact that RoPs of adjacent nodes in ofmap are overlapped, different ifmap streams contain a significant amount of repeated data.

We take advantage of the properties of convolution layer in Section II and propose the vertical data-streaming technique in order to avoid repeat streaming of same data in ifmaps. The technique consists of the in-depth addressing mode and vertical streaming architecture, which are illustrated in this section.

A. In-depth addressing mode

The conventional addressing format of ifmaps in memory is shown in Figure 2. For a three-channel RGB ifmaps, the blue ifmap is first stored in memory, following by green and red ifmaps separately. Such horizontal addressing mode follows the perception-behavior of graphical data for human. However, such addressing mode leads to repeated streaming of same ifmap data, which is mainly caused by overlapping RoP for adjacent nodes in ofmaps as previously explained. Alternately, we propose to store the ifmaps according to Figure 3, where nodes at the same position across all ifmaps are connected in the memory location. Besides ifmaps and ofmaps, such in-depth addressing mode can

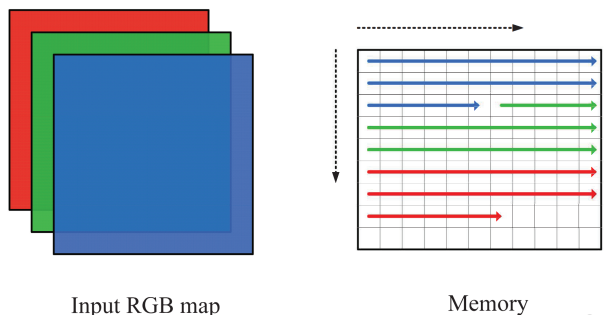

Fig. 2: Horizontal addressing mode of ifmaps and ofmaps

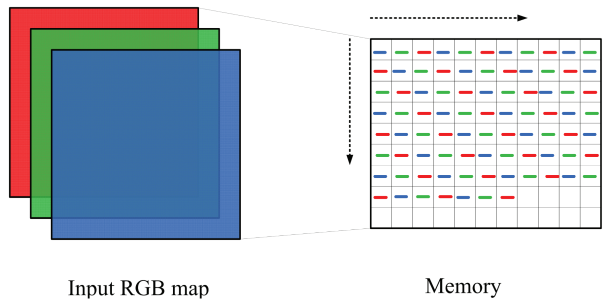

Fig. 3: In-depth addressing mode of ifmaps and ofmaps

be also applied to parameters of the neural networks, such as convolution filters.

The computing of convolution layer using in-depth addressing mode is illustrated in Figure 4. When in-depth addressed nodes in ifmaps are streamed-in from memory, they are identically demanded by all in-depth addressed nodes in ofmaps. Using the in-depth addressed filters, the weighted addition between data and parameters for convolution remains the same as horizontal addressing mode. The resulted ofmaps in in-depth mode can be directly streamed-in as the ifmaps of the next network layer without additional data processing.

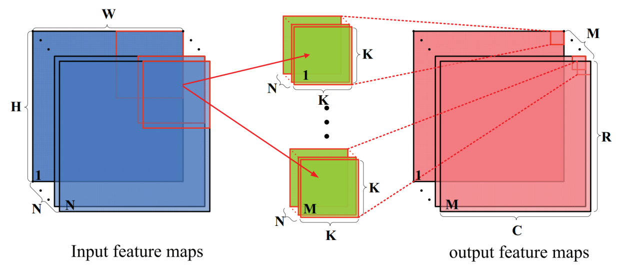

Fig. 4: In-depth addressing based computing for convolution

B. Vertical streaming architecture

The proposed vertical data streaming architecture is present in Figure 5. Two memory blocks (SRAM or DRAM) contain data (ifmaps/ofmaps) and network parameters (filters/weights), an array of PEs are implemented on-chip which mainly contains multiplier, adder and SRAM block. In-depth addressed data is streamed-in and streamed-out through shift registers, while parameters are streamed-in to fill the SRAM for each PE.

To maximally share the streamed-in data, each PE processes individual channel of ofmap and only stores filters for the specific ofmap by in-depth addressing mode. The output data streams are stored temporarily in the data memory, waiting for streaming-in for the next network layer.

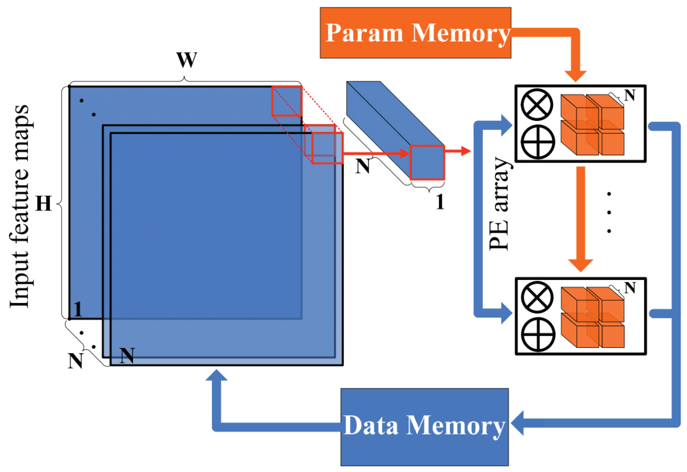

Fig. 5: Vertical data-streaming architecture for neural network

a) Configurable knobs: The architecture can be configured dynamically. Configuration knobs include dimensions of each ifmaps and ofmaps, size of convolution filter, whether padding is required and size of stride. To configure the architecture, one shift register of 96-bit is used for programming operating modes. Besides, the memory location for storing parameters is also defined through the shift register.

b) Address generator: The address generator for ifmaps and ofmaps ensures data streams are fetched from and written to correct memory location. Under vertical streaming mode, Ifmaps can be sequentially streamed-in and shared by multiple PEs. Ofmaps streams need to be occasionally interleaved in writing location due to the insufficient amount of PEs to complete all output channels in parallel, however, the design of output address generator follows simple prior knowledge on network structure. In contrast, under horizontal streaming mode, the input data stream cannot be shared by all PEs, so that each PE needs to be equipped with additional logic to acquire correct data based on individual RoP. Extra requirements such as padding insertion can be extremely difficult to implement by horizontal streaming but greatly simplified under vertical streaming mode.

c) Support for other computing layers: Vertical streaming not only favors the convolution layer but is also applicable to pooling, activation, fully connected and even Local Region Normalization (LRN) layer where adjacent data from several channels are normalized. Mode controlling logic is implemented on address generators and PEs in order to switch between different operating modes. For computing of different layers, the same set of hardware modules is shared to avoid the implementation with multiple copies of algorithmic units in the same PE.

C. Discussion

The main advantage of the proposed design is its high PE utilization. Since all PEs identically demand the same input data stream for convolution, the PE utilization rate approximates 100%. We clock gate the PEs during the gap between the current streaming-out and the next streaming-in phase in order to save additional dynamic power consumption.

The default architecture has 128 PEs which process maximally 128 channels of ofmaps in parallel. For the state-of-the-art CNN networks, each layer has more channels of ofmaps than the amount of PEs. Hence repeated streaming-in of data is only required in this case. Figure 6 shows the averaged amount of feature map channels for several state-of-the-art CNN structures, it is observed that the largest Googlenet has an average channel count of 512, which demands 4 batches of input data streaming under current architecture configuration.

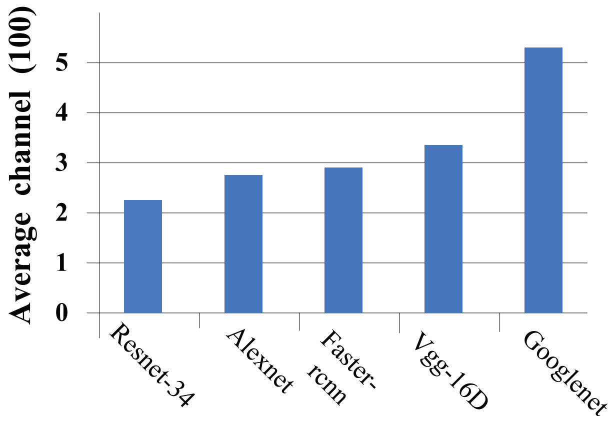

Fig. 6: Average amount of feature map channels for mainstream CNN structures

Besides PE utilization, the proposed architecture keeps a relatively small amount of SRAM (6K for each PE) instead of sharing a large pool of SRAMs. Such a design avoids the contention due to concurrent fetch of filter weights, hence facilitates parallel processing. For the sake of simplicity in control logic, the parameters are streamed-in after the streaming-out of the current batch of ofmaps. Therefore both data and parameter streams can share the same memory such as area-friendly single port DRAM.

IV. Experimental results

A. Design methodology

The vertical data-streaming architecture is designed by Verilog and simulated using Modelsim. The adopted physical design phases include logic synthesis with design compiler and placement and route through Cadence EDI. The design is synthesized at 100MHz using 65nm CMOS standard cell library and SRAM IPs. Simulation memories are used for storing both network data and parameters.

B. Performance profiling

Convolution layers of VGG-16 networks are deployed onto the streaming architecture, where the simulated performance for processing a batch of three RGB pictures are shown in Table I. The total latency includes the processing latency and latency for loading parameters. MAC counts both multiply and addition as one operation. The number of RAM access shows the amount of data streamed from and to the data memory.

TABLE I: Benchmark of performance for 13 convolution layers in VGG-16 (Batch size equals three) with Eyeriss [10]

Layer	**Total Latency (ms)**	**Total Latency of Eyeriss (ms)**	**Processing Latency(ms)**	**Number of MACs**	**Input fmap channel**	**Num of Active PEs**	**Num of Active PEs of Eyeriss**	**DRAM Accesses**
CONV1-1	136.9	76.2	40.6	0.26G	64	64(50%)	156(93%)	13MB
CONV1-2	963.3	910.3	867	5.55G	64	64(50%)	156(93%)	91.8MB
CONV2-1	264.9	470.3	216	2.77G	128	128(100%)	156(93%)	25.2MB
CONV2-2	481.7	894.3	433.5	5.55G	128	128(100%)	156(93%)	45.9MB
CONV3-1	240.8	241.1	216.7	2.77G	256	128(100%)	156(93%)	22.9MB
CONV3-2	457.6	460.9	433.5	5.55G	256	128(100%)	156(93%)	43.6MB
CONV3-3	457.6	457.7	433.5	5.55G	256	128(100%)	156(93%)	43.6MB
CONV4-1	228.8	135.8	216.7	2.77G	512	128(100%)	168(100%)	21.8MB
CONV4-2	445.6	254.8	433.5	5.55G	512	128(100%)	168(100%)	42.4MB
CONV4-3	445.6	246.3	433.5	5.55G	512	128(100%)	168(100%)	42.4MB
CONV5-1	111.4	54.3	108.3	1.39G	512	128(100%)	168(100%)	10.6MB
CONV5-2	111.4	53.7	108.3	1.39G	512	128(100%)	168(100%)	10.6MB
CONV5-3	111.4	53.7	108.3	1.39G	512	128(100%)	168(100%)	10.6MB
Total	**4457.5**	**4309.5**	**4050.7**	**46.04G**	**325**	**118(92%)**	**158(94%)**	**425.1MB**

The proposed design achieves an average of 92% active PEs during the data streaming phase while for the rest cycles the PEs are clock-gated. Since 128 PEs are implemented in the architecture, CONV1-1 and CONV1-2 layers have only 64 channels of feature map which leads to the percentage of active PEs as 50%. In term of throughput, our design at 100 MHz complete all convolution layers at 0.67 frames per second. Table I also shows the comparison with Eyeriss [10] for total latency and amount of active PEs. It is shown that for both design metrics our proposed architecture achieves similar performance.

C. Physical results

Table II shows the physical properties of proposed engine while Figure 7 presents the view of post placement netlist.

TABLE II: Physical properties of CNN streaming engine

Technology	$65nm$ UMC CMOS
Chip area	$11.8mm^2$
Power estimate (DC)	$20.035mW@1.08V$
Total SRAM size	$769K$ bytes
Configuration SRAM size	$1K$ bytes
SRAM size per PE	$6K$ bytes
Number of PEs	128
Clock frequency	$100MHz$
Configurable features	Filter size: $1 \times 1 to 32 \times 32$ #. of filters: $1 to 1024$ #. of ifmaps: $1 to 1024$ #. of ofmaps: $1 to 1024$ Require padding? (Y/N)
Current supported layers	Convolution, pooling, full connection

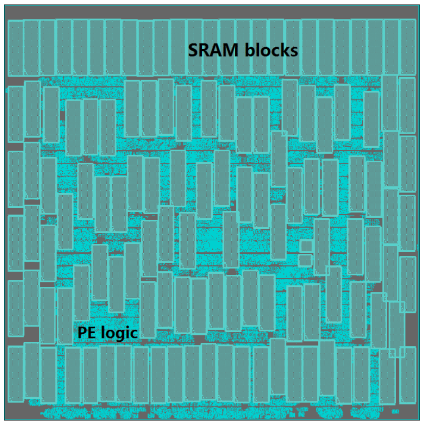

Fig. 7: Physical placement for the CNN streaming engine

V. Conclusion

In this work, a vertical data-streaming architecture for a convolutional neural network is present by introducing the in-depth addressing mode of both network data and parameters. The design achieves high resource utilization by maximally sharing of input data-stream across multiple output feature maps. The simulation and physical results demonstrate that the design based on 65nm CMOS technology achieves 0.67 frames per second for all convolution layers of VGG-16 network with a small power estimate of 20.035 mW.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, p. 436, 2015.
[2] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
[3] R. Girshick, “Fast r-cnn,” in *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448, 2015.
[4] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” in *Advances in neural information processing systems*, pp. 91–99, 2015.
[5] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask r-cnn,” in *Computer Vision (ICCV), 2017 IEEE International Conference on*, pp. 2980–2988, IEEE, 2017.
[6] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.
[7] F. Yu and V. Koltun, “Multi-scale context aggregation by dilated convolutions,” *arXiv preprint arXiv:1511.07122*, 2015.
[8] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, “Mastering the game of go with deep neural networks and tree search,” *nature*, vol. 529, no. 7587, p. 484, 2016.
[9] J. Cong and B. Xiao, “Minimizing computation in convolutional neural networks,” in *International conference on artificial neural networks*, pp. 281–290, Springer, 2014.
[10] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, “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.
[11] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, *et al.*, “In-datacenter performance analysis of a tensor processing unit,” in *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*, pp. 1–12, IEEE, 2017.
[12] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, *et al.*, “Dadiannao: A machine-learning supercomputer,” in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 609–622, IEEE Computer Society, 2014.
[13] Y.-H. Chen, J. Emer, and V. Sze, “Using dataflow to optimize energy efficiency of deep neural network accelerators,” *IEEE Micro*, vol. 37, no. 3, pp. 12–21, 2017.
[14] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, “Shidiannao: Shifting vision processing closer to the sensor,” in *ACM SIGARCH Computer Architecture News*, vol. 43, pp. 92–104, ACM, 2015.