

VSCNN: Convolution Neural Network Accelerator With Vector Sparsity

Kuo-Wei, Chang, and Tian-Sheuan Chang

Dept. of Electronics Engineering, National Chiao Tung University Hsinchu, Taiwan

Abstract—Hardware accelerator for convolution neural network (CNNs) enables real time applications of artificial intelligence technology. However, most of the accelerators only support dense CNN computations or suffers complex control to support fine grained sparse networks. To solve above problem, this paper presents an efficient CNN accelerator with 1-D vector broadcasted input to support both dense network as well as vector sparse network with the same hardware and low overhead. The presented design achieves 1.93X speedup over the dense CNN computations.

Index Terms—Hardware design, convolution neural networks (CNNs), sparse CNNs.

I. INTRODUCTION

Convolution neural networks (CNN) have been widely used in computer vision such as recognition [1]–[5], detection [6]–[10], and autonomous vehicles during recent years for its significant improvement over traditional approaches. However, computations of CNNs demands a lot of multiplications and accumulations (MACs), and millions of data amount per layer. Thus, hardware accelerators for CNNs are required to meet real time applications.

Various hardware accelerators have been proposed recently [11]–[16], which can be divided into dense CNN or sparse CNN computation types. The dense CNN types [11]–[14] assume continuous and regular computational data flow to the hardware accelerator, which results in simple and regular systolic array or filter-like architecture. However, CNN computations contain a lot of zeros in its weight and input activations due to model pruning [17] and popular ReLU activation function. Exploring sparsity offers a significant speedup option of the hardware accelerator. But this type of sparsity is a fine grained sparse structure as shown in Fig. 1, where the zero distribution is abundant but irregular and bad for hardware design. To achieve sparse CNN computation, [11] tried the gated input for zero input with the same dense CNN design to save power, which did not save computation cycles. [15] skipped zero weight computation on its single instruction multiple data (SIMD) array by the zero weight indexing and their distance. [16] explored both zero weight and input with a nonzero data indexing system, computed them by a 2D multiplier array, and accumulated those sparse outputs with the help of coordinate computation to sort these irregular output.

K. Chang and T. Chang, "VSCNN: Convolution Neural Network Accelerator with Vector Sparsity," 2019 IEEE International Symposium on Circuits and Systems (ISCAS), 2019, pp. 1-5, doi: 10.1109/ISCAS.2019.8702471.

All these designs [11], [15], [16] are for the fine grained sparsity. The irregularity of the fine grained sparsity results in significant area cost on the indexing system and data routing.

To reduce above cost while still explore the benefit of sparsity, this papers proposes a CNN accelerator to support dense CNN computation as well as vector sparse CNN on both weight and input. The vector sparsity as shown in Fig. 2 [18] has vectors of zeros instead of fine grained ones, which enables regular hardware design and still offers zero skipping benefits as shown in our experimental results. To support this, the proposed design operates on a 1-D to 1-D matrix multiplication with broadcasted 1-D weight and input, which enables zero vector skipping easily.

This CNN accelerator design has the following contributions

- Support dense CNN, and vector sparse CNN in one design with the same accumulator flow and an index system for vector sparsity.
- Skip both zero weight data and input data to get great performance.

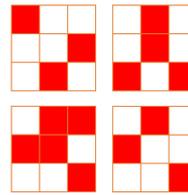


Fig. 1. Fine grained sparse structure

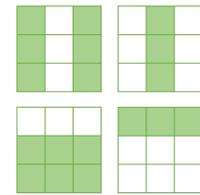


Fig. 2. Vector sparse structure

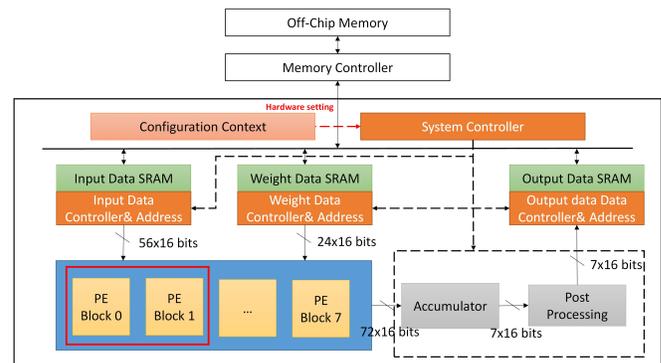


Fig. 3. The proposed system architecture

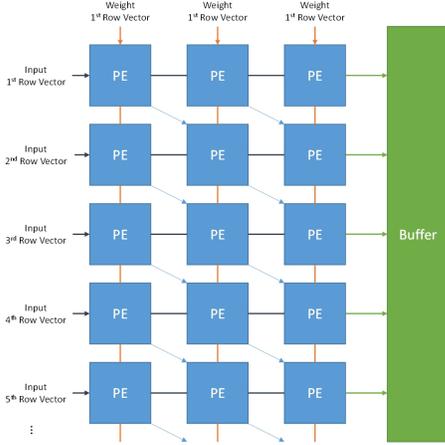


Fig. 4. The PE array architecture

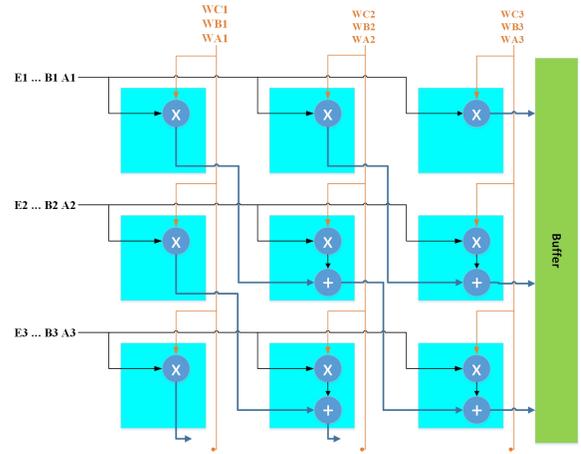


Fig. 5. The overview of processing element

The rest of the paper is organized as following. Section II shows the overview of our proposed architecture. Section III gives detailed data flow for dense CNN and vector sparse CNN. The implementation results and comparison are shown in section IV. Finally, we conclude in section V.

II. ARCHITECTURE

A. Overview

Fig. 3 shows the proposed system architecture. This design first gets the input and weight data from external memory and stores them into the local SRAM buffers for the following repeated access. These data are fed into the processing element (PE) array to compute convolution and then accumulated through the accumulator according to the index. During the accumulation, these partial sum of the convolution results are stored in a local SRAM buffer to avoid unnecessary external memory access until the final accumulated output is generated. These accumulated output is processed by the post processing unit for following activation functions, normalization, and zero detection. The final output which are non-zero vector will be sent back to external DRAM. The whole process is controlled by the system controller according to the configuration context. The data access of input, weight and output are controlled by their SRAM buffer controllers to sequentially accessing the data.

B. PE array

The PE array is the main processing core of this proposed design as shown in Fig. 4 and Fig. 5. Each PE as in Fig. 5 contains one multiplier to multiply input and weight, and adder

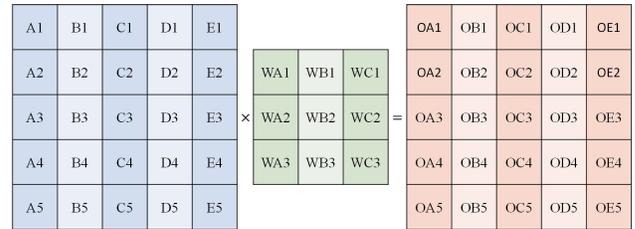


Fig. 6. An convolution example with 5x5 input with padding 1 and 3x3 weight to generate 5x5 output

for partial sum accumulation. In this design, the weights are not stored locally in each PE as in other designs since such design style is not suitable for sparse computation. Instead, the input data are broadcasted horizontally and the weight data are broadcasted vertically to support both dense and vector sparse computation. The partial results are then propagated diagonally along the PE array, as shown in Fig. 5 to accumulate in the same cycle.

The hardware utilization of the PE array depends on how to map the kernels and inputs to the array and the sparsity of the network. Since the 3x3 convolution is the most widely filter in current CNNs, our architecture has optimized the convolution process for 3x3 filters with the unit stride for full hardware utilization. Other filter sizes and non-unit strides can be supported as well by a suitable mapping method [13].

III. DENSE AND SPARSE DATA FLOW

Fig. 7 shows the data flow of the CNN computation with 15 PEs for a 5x5 input with padding 1, and 3x3 filter kernel example as in Fig. 6. At the first cycle, the first column of the weight filter, WA1 to WA3, is broadcasted vertically to the PE array. The corresponding first column of the input activations, A1 to A5, is also broadcasted horizontally to the PE array. The vector to vector multiplication results are also summed together along the diagonal direction at the same cycle to generate part of the results of OB1 to OB5, as illustrated in Table. I and Fig. 8. These partial results (e.g. OB1 to OB5 at $t = 1$) are stored in the buffer and accumulated with the next

TABLE I
TIMING DIAGRAM

Dense CNN Timing Diagram										
Cycle	1	2	3	4	5	6	7	8	9	...
Input	A1-A5			B1-B5			C1-C5			...
Weight	WA1-WA3	WB1-WB3	WC1-WC3	WA1-WA3	WB1-WB3	WC1-WC3	WA1-WA3	WB1-WB3	WC1-WC3	...
Output	OB1-OB5	OA1-OA5	x	OC1-OC5	OB1-OB5	OA1-OA5	OD1-OD5	OC1-OC5	OB1-OB5	...
Sparse CNN Timing Diagram										
Cycle	1	2	3	4	5	6	7	8	...	
Input	A1-A5		C1-C5		D1-D5		E1-E5		...	
Weight	WA1-WA3	WB1-WB3	WA1-WA3	WB1-WB3	WA1-WA3	WB1-WB3	WA1-WA3	WB1-WB3	...	
Output	OB1-OB5	OA1-OA5	OC1-OC5	OB1-OB5	OE1-OE5	OD1-OD5	x	OE1-E5	...	

partial results with the same index (e.g. OB1 to OB5 at $t = 5, 9$).

For dense CNN computation as in Fig. 8, each output will take 3 different cycles for 3×3 filters, and 15 cycles for 5×5 input. For sparse CNN computation, zero input data and weight data as denoted by the dashed line block will not be in SRAM, so they will be skipped and not be computed as shown in Table. I and Fig. 8. Only the nonzero part will be in SRAM and sent to the PE array and accumulated with the same index system. Thus, the computation cycles are reduced (e.g. $t=3, 4, 5, 6$ and 9 in Fig. 8) while still keep the computation regular. As result, we only need 8 cycles for this sparse CNN, saving 47% of cycles.

IV. EXPERIMENTAL RESULT

The proposed architecture has been implemented and simulated with the VGG-16 model pretrained on the ImageNet dataset and pruned with the vector pruning method as [18]. The accuracy only drop 0.08% with density 23.5%. The PE number used in the simulation is 168, arranged in two configurations: $[4, 14, 3]$ 4 PE arrays, and 14 rows and 3 columns per PE array, $[8, 7, 3]$ 8 PE arrays, and 7 row and 3 columns per PE array. Such configurations are chosen to maximize the hardware utilization for above VGG-16 model execution. With above configuration, the input activation vector size is set to

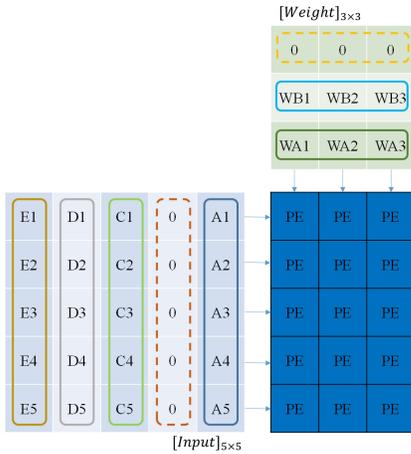


Fig. 7. Illustration of data flow (dashed line block represents all zero vector in sparse CNN)

14 or 7. Following output zero detection in post processing element and weight pruning, Fig. 9 and Fig. 10 shows the nonzero data density of input activation and weight for fine grained sparsity and vector sparsity, respectively. As expected, the fine grained sparsity has lower density than that in the vector sparsity case.

The speedup results of the proposed design are shown in Fig. 12 and Fig. 13. When compared with the dense CNN computation, we can achieve 1.871X and 1.93X speedup for $[4, 14, 3]$ and $[8, 7, 3]$ cases, respectively. Small number of PE rows in the $[8, 7, 3]$ case results in more zero vectors to skip, and thus higher speedup, but the difference is small. The zero computation that this design can skip is 92% ($[4, 14, 3]$ case) and 85% ($[8, 7, 3]$) compared to their respective ideal vector sparse computation, and 46.6% ($[4, 14, 3]$ case) and 47.1% ($[8, 7, 3]$) compared to the ideal fine grained computation. Our design is efficient to exploit almost all zero vectors. Small zero vector enables more zero skipping.

When compared to the fine grained design in [16], our design overhead is very small compared to the complex index, accumulator and routing in [16]. The speedup over the dense CNN in [16] is about 3X, which roughly exploits 66% of ideal fine grained zero computation. In comparison, our design can exploits 47% in average of ideal fine grained zero computation to achieve 1.93X speedup with small area overhead. Our design is more hardware efficient than the previous design.

V. CONCLUSION

This paper proposes a CNN hardware accelerator that can support dense CNN and vector sparse CNN incurring small area overhead with broadcasted 1-D input activation vector and 1-D weight vector data flow. This design can achieve 1.93X speedup over the dense CNN computation by exploiting around 90% of the ideal zero vector computation.

ACKNOWLEDGMENT

This work was supported by Ministry of Science and Technology, Taiwan, under Grant 108-2634-F-009 -005, 107-2119-M-009-019, and Research of Excellence program 106-2633-E-009-001.

REFERENCES

- [1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in Proc. NIPS, 2012, pp. 1097-1105.

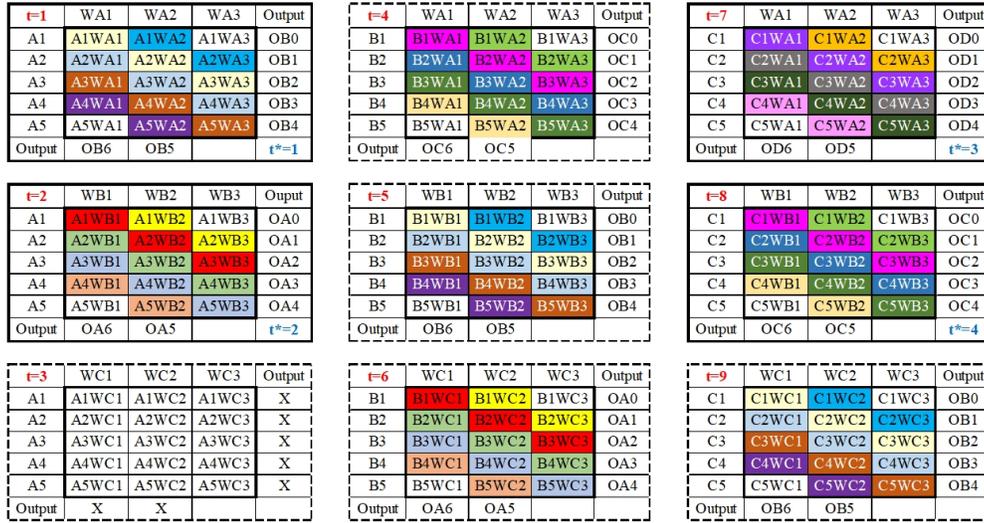


Fig. 8. Dataflow chart for dense and sparse CNN computation in Fig. 8, where the dashed line blocks will be skipped at the sparse CNN case. In each block, the element with the same color will be summed together in a PE. OA0, OA6, OB0, OB6, ... are for zero padding boundary computation. t^* represents cycles for sparse CNN.

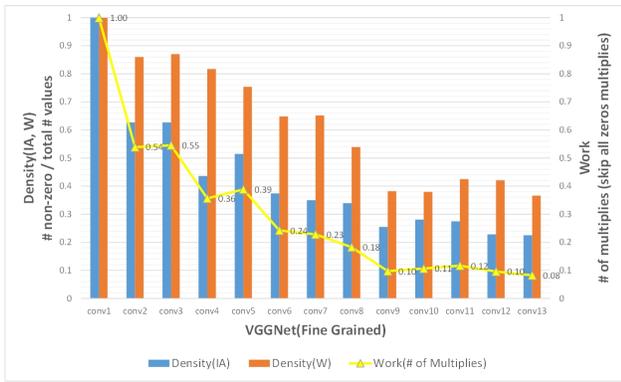


Fig. 9. Density ratio of input and weight and work with fine grained sparse

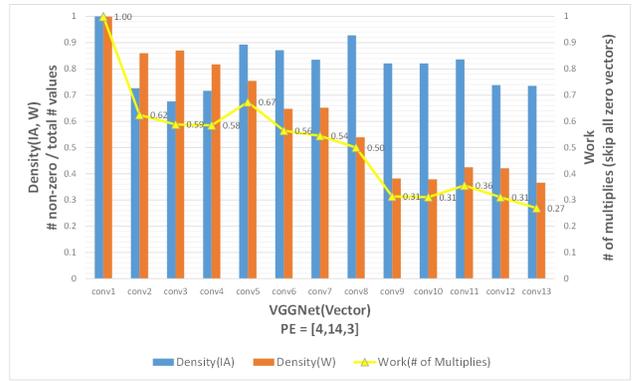


Fig. 10. Density ratio of input and weight and work with PE array 4 blocks, 14 rows, 3 columns

- [2] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, et al., "Going deeper with convolutions," in Proc. IEEE CVPR, 2015, pp. 1-9.
- [3] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," CoRR, 2014..
- [4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proc. IEEE CVPR, 2016.
- [5] S. Ji, W. Xu, M. Yang, and K. Yu, "3D convolutional neural networks for human action recognition," IEEE Trans. Pattern Anal. Mach. Intell., vol. 35, pp. 221- 231, 2013.
- [6] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: integrated recognition, localization and detection using convolutional networks," in Proc. ICLR, 2014.
- [7] R. Girshick, J. Donahue, T. Darrell, and J. Malik, Rich feature hierarchies for accurate object detection nd semantic segmentation," in Proc. IEEE CVPR, 014, pp. 580-587.
- [8] T. He, W. Huang, Y. Qiao, and J. Yao, "Textattentional onvolutional neural network for scene text etection," IEEE Trans. Image Process., vol. 25, pp. 529-2541, 2016.
- [9] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua, "A onvolutional neural network cascade for face etection," in Proc. IEEE CVPR, 2015, pp. 5325-5334.
- [10] Tomè, F. Monti, L. Baroffio, L. Bondi, M. Tagliasacchi, and S. Tubaro, "Deep convolutional neural networks for pedestrian detection," Signal

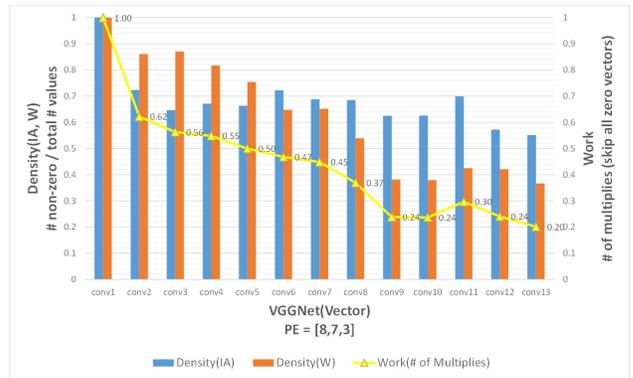


Fig. 11. Density ratio of input and weight and work with PE array 8 blocks, 7 rows, 3 columns

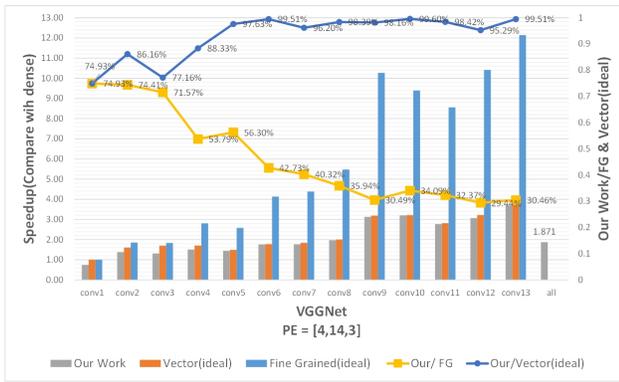


Fig. 12. Speedup of our work and ideal vector sparse and fine grained sparse network with PE array 4 blocks, 14 rows, 3 columns.

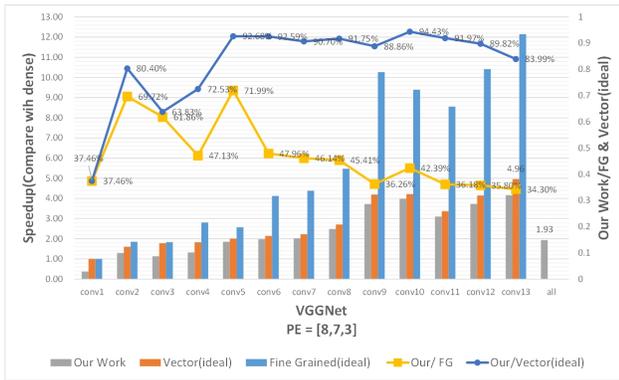


Fig. 13. Speedup of our work and ideal vector sparse and fine grained sparse network with PE array 8 blocks, 7 rows, 3 columns.

Processing: Image Communication, 2016.

- [11] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," in IEEE ISSCC, 2016
- [12] F. Tu, S. Yin, P. Ouyang, 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, vol. 25, no. 8, pp. 2220–2233, 2017
- [13] Y.-J. Lin and T. S. Chang, "Data and hardware efficient design for convolutional neural network," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 65, no. 5, pp. 1642–1651, May 2018.
- [14] S. Liu et al., "Cambricon: An instruction set architecture for neural networks," in Proc. ISCA, 2016.
- [15] S. Zhang et al., "Cambricon-X: An accelerator for sparse neural networks," in Proc. Annu. Int. Symp. Microarchitecture (MICRO), Oct. 2016, pp. 1–12.
- [16] A. Parashar, et al., "SCNN: An accelerator for compressed-sparse convolutional neural networks," in Proc. ISCA, 2017, pp. 27–40.
- [17] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding," in ICLR, 2016.
- [18] H. Mao, et al., "Exploring the regularity of sparse structure in convolutional neural networks," in Proc. CVPR Workshop Tensor Methods In Comput. Vis., 2017.