



Hardware Accelerators for Deep Learning Focusing on Sparsity and Computing-in-Memory

Dingchen Zhu

College of College of Integrated Circuits and Micro-Nano Electronics, Fudan University,
Shanghai, China

22307130071@m.fudan.edu.cn

Abstract. This paper presents a comprehensive review of hardware accelerators specifically designed to address the challenges posed by sparse neural networks, focusing on their architectural innovations, algorithmic optimizations, and performance advancements. Traditional computing systems face significant challenges such as energy inefficiency, memory bottlenecks, and computational irregularities due to the inherent complexity and sparsity of modern neural networks. To overcome these issues, this paper analyzes five representative hardware accelerators used in deep learning computations: Eyeriss, Cambricon-S, NullHop, Tensaurus, and GAS. These architectures employ various strategies, including dataflow optimization, sparsity-aware pruning, zero-skipping mechanisms, and in-memory computing, to achieve significant performance gains. The results demonstrate notable improvements, such as $2.5\times$ energy efficiency in Eyeriss, $1.37\times$ efficiency gains in Cambricon-S, and 3 TOP/s/W power efficiency in NullHop, along with breakthroughs in handling mixed sparse-dense computations in Tensaurus and the introduction of a unified in-memory architecture in GAS. This analysis highlights the critical role of hardware-software co-design and effective sparsity exploitation in enabling scalable, energy-efficient AI systems. By consolidating key findings and strategies, this work provides a roadmap for future research aimed at optimizing neural network acceleration for real-world applications.

Keywords: Sparse Weight; Hardware Accelerator; Neural Network Optimization

1 Introduction

Recent innovations in specialized hardware accelerators aim to bridge this gap by co-designing algorithms and architectures tailored to sparse computations [1-3]. This paper systematically examines five pivotal accelerators that exemplify diverse approaches to sparsity optimization [4,5]. Eyeriss introduces a row-stationary dataflow to minimize DRAM access, achieving $1.4\text{--}2.5\times$ energy efficiency gains in CNNs [6]. Cambricon-S leverages coarse-grained pruning and hardware selectors to compress sparse networks by $79\times$ without accuracy loss [7]. NullHop exploits activation sparsity through zero-

skipping, achieving 98% MAC utilization and 450 GOp/s throughput [8]. Tensaurus unifies sparse and dense tensor processing, delivering 22.9× speedup over CPUs for factorization tasks [9]. GAS integrates in-memory computing with outer-product-based sparse matrix multiplication, outperforming GPUs in energy efficiency [10]. This analysis reveals critical design principles: reducing data movement through localized computation, leveraging sparsity-aware algorithms to eliminate redundant operations, and integrating memory and processing units for parallelism. This work not only benchmarks the state-of-the-art but also provides insights into emerging trends, such as hybrid sparse-dense support and flexible in-memory architectures. The findings underscore the importance of cross-disciplinary collaboration between algorithm designers and hardware engineers to unlock the full potential of sparse neural networks in real-time, resource-constrained environments.

2 Key Concepts

Convolutional Neural Networks (CNNs), as the most essential architecture in deep learning, have demonstrated outstanding performance in fields such as image recognition, video analysis, and natural language processing due to their unique local connectivity and weight-sharing mechanisms [11-13]. A traditional CNN model is comprised of convolutional layers, pooling layers, and fully connected layers, where convolution operations extract local features from input feature maps using a sliding window, progressively forming high-level semantic representations [14]. However, the computational characteristics of CNNs—high-dimensional tensor operations, repetitive multiply-accumulate (MAC) computations, and frequent transmission of large intermediate data—pose significant challenges when running on traditional Von Neumann architectures. These architectures suffer from severe “memory wall” and “power wall” issues. For instance, a single inference of ResNet-50 requires approximately 3.8×10^9 floating-point operations, yet only 20% of the energy is utilized for actual computation, while the remaining 80% is consumed primarily in data movement. To overcome these limitations, dedicated neural network hardware accelerators have emerged, leveraging architectural innovations to achieve coordinated optimization of computation, storage, and communication resources.

2.1 CNN Basis

Convolutional Neural Networks (CNNs for short) are models designed for processing data in grid shape, commonly used to represent images. They are comprised of multiple layers, such as the convolutional layers, activation functions (e.g. ReLU function), pooling layers, and fully connected layers. The convolutional layers (shown in Figure 1) extract hierarchical features using weight kernels, while pooling layers reduce spatial dimensions to improve computational efficiency and generalization. CNNs have been widely used in image recognition, object detection, and medical image analysis due to their ability to automatically learn feature representations.

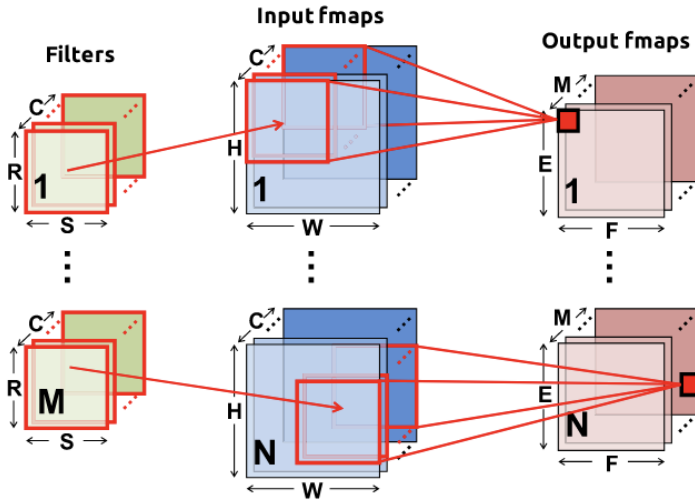


Fig. 1. The computation in cnn layer which involves the filters, input fmaps(feature maps) and output fmaps [6].

A critical challenge in CNN computation is the intensive data movement between the main memory and processing elements, as well as the high computational cost of convolution operations. This motivates the need for hardware accelerators to optimize CNN execution.

2.2 Hardware Accelerator

Due to the computational complexity of CNNs, hardware accelerators have been developed to improve efficiency. These include FPGAs, ASICs, and GPUs [6, 15, 16]. Each accelerator has unique trade-offs: GPUs: High throughput and flexibility, but suffer from high power consumption. FPGAs: Reconfigurable and power-efficient but may have lower peak performance. ASICs: Custom-designed for optimal efficiency and performance but lack flexibility. State-of-the-art (SOTA) CNN accelerators, such as Eyeriss, optimize memory access using dataflow architectures like row-stationary processing, reducing data movement costs. Other accelerators, like Tensaurus and NullHop, leverage sparsity in neural networks, skipping computations on zero values to improve efficiency. The combination of algorithmic sparsity and hardware specialization has become a dominant strategy in modern neural network accelerators, enabling faster and more power-efficient deep learning inference.

3 Hardware Accelerators For Sparse Deep Learning And Computing-In-Memory

Efficient hardware acceleration for neural networks has been extensively studied, with various architectures designed to optimize computation and memory access. In this paper, this paper concludes the related works and categorizes these models into four types: Dataflow Optimization, Pruning and Regularization, Sparse Computation Support and Compute-in Memory.

3.1 Early Dataflow Optimization: Eyeriss

The Eyeriss focuses on a highly energy-efficient, reconfigurable accelerator for the computation in deep CNNs, addressing the challenges posed by data movement in CNNs, which often consumes more energy than computation itself. Eyeriss employs a Row-Stationary (RS) dataflow, which optimizes the data movement between main memory and the processing elements (PEs) by making each PE process data stored locally. This algorithm minimizes expensive accesses to off-chip DRAM and maximizes the reuse of data in the low-cost memory hierarchy (scratch pads and inter-PE communication). The architecture is a spatial array of 168 PEs, as shown in Figure 2, and it is organized into a 12x14 grid, with a global buffer (GLB) and a network-on-chip (NoC). It integrates four levels of memory, each with varying energy costs: DRAM, GLB, inter-PE communication, and local scratch pads.

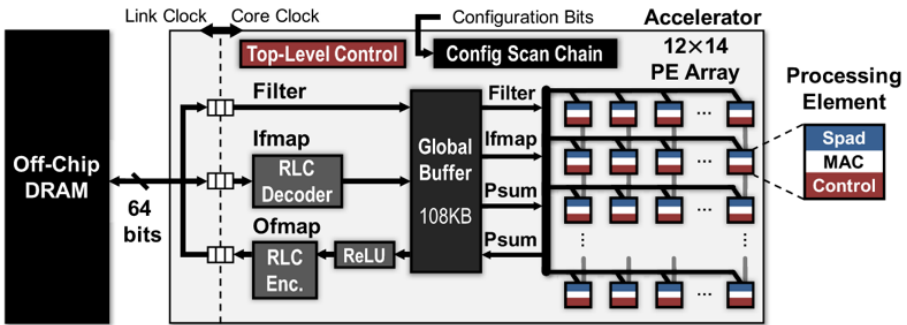


Fig. 2. The overall architecture of eyeriss, which includes glb, noc and other essential peripheral elements of the computing circuit [7].

The RS dataflow works by splitting convolutions into 1D primitives, each of which processes a single row of filter weights and a single row of input feature map (ifmap) values. These rows are processed in parallel within the PE array, minimizing memory accesses. Each PE can handle different parts of the computation independently, which improves the accelerator's throughput and efficiency. Eyeriss shows significant energy improvements over previous CNN accelerators. It achieves 1.4–2.5× better energy efficiency in AlexNet and other state-of-the-art CNNs, processing at 35 frames per

second with very low DRAM accesses, making it a highly efficient solution for large-scale CNN applications.

3.2 Sparsity-Aware Pruning and Regularization: Cambricon-S, NullHop

To cater for the exponential growth in computation for AI, the accelerators are designed with deeper architectures, which pose a big challenge to the management of data volume and computational requirements. To minimize the cost caused by the data movement, especially the sparse data, the two papers below propose a synergetic method, which link the hardware and the software, aiming to explore a better way to design the CNN accelerator. The two works combine the hardware accelerator with pruning and regularization, respectively. The Cambricon-S accelerator uses a cooperative software/hardware approach to address the irregularity introduced by sparsity in neural networks. The software employs coarse-grained pruning to reduce irregularity by grouping and pruning clusters of synapses, which significantly improves data compression. Additionally, local quantization is applied to reduce the size of weight data. Cambricon-S features a hardware accelerator with a selector module designed to filter out redundant neurons and synapses, thereby improving computational efficiency and reducing memory overhead.

Local convergence occurs during training, where larger weights tend to cluster, as observed in the study. This observation is leveraged for coarse-grained pruning, which reduces sparse irregularity and improves efficiency in both the software and hardware level. A neuron selector module and several synapse selector modules are employed in the hardware to eliminate redundant data during computation.

Compared to existing SOTA neural network accelerators for sparsity, Cambricon-S provides $1.71\times$ better performance and $1.37\times$ better energy efficiency. The hardware is capable of handling sparse neural networks with up to $79\times$ compression without significant accuracy loss, demonstrating the approach's effectiveness in improving performance and reducing energy consumption. Since the previous works lack flexibility and reconfigurability, the following work, NullHop, proposes a more efficient CNN accelerator to enable both low-power operation and real-time performance. NullHop uses activation sparsity to optimize the CNN computation. The architecture features a zero-skipping mechanism that avoids unnecessary operations on zero activations, which makes the computation more efficient. Additionally, it adopts an algorithm to compress the sparse matrix, which combines a sparsity map (SM) and a non-zero value list (NZVL) to reduce memory requirements and efficiently manage data flow through the accelerator.

As illustrated in Figure 3, the accelerator is designed with 128 MAC units, supporting convolutional layers, ReLU activation, and max-pooling operations. Data is processed by an Input Decoding Processor (IDP) that decodes compressed feature maps, then a Compute Core Module (CCM) performs the convolutional calculations. The system also includes flexible kernel sizes (1x1 to 7x7) and has a max capacity up to 128 input and output feature maps per layer.

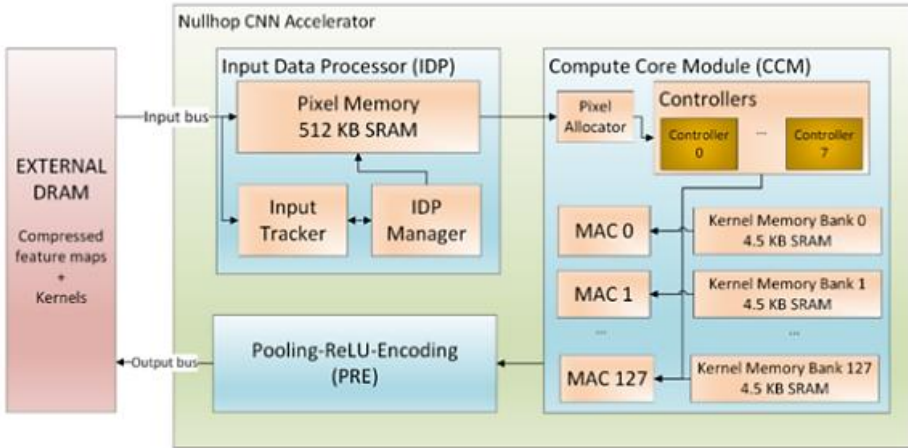


Fig. 3. The architecture of nullhop [8].

The system firstly decodes input feature maps in a compressed form, which includes the application of zero-skipping and the reduction of redundant MAC operations. Non-zero values are processed, with their corresponding pixel positions mapped from the SM, ensuring efficient data handling. These operations are conducted in parallel across MAC units for higher throughput. The NullHop is implemented on an FPGA (Xilinx Zynq) and achieves up to 450 GOP/s with a power efficiency of 3 TOP/s/W and 98% MAC utilization. In the test, it could process large networks like VGG19 with an extremely high efficiency of 368%, indicating that the architecture can handle sparsity without performance loss. The use of compression reduces external memory access, which lowers the power consumption and runtime for real-time applications.

3.3 Hybrid Sparse-Dense Computation Support: Tensaurus

The tensor factorizations are commonly seen in most of the machine learning and data analytics works. Since tensors often exhibit sparsity in real-world applications, this paper proposes a hardware accelerator to efficiently handle both dense and sparse tensor factorizations. Tensaurus introduces a hardware accelerator optimized for both sparse and dense computations in tensor. The core computation pattern, SF3, is observed across multiple tensor factorization kernels and matrix operations. The process includes scalar-vector multiplications, element-wise vector operations, and fiber accumulation, which is efficiently executed using SIMD (Single Instruction, Multiple Data) across multiple processing elements (PEs). The system reads out sparse tensor data in CISS format, which allows it to maximize memory bandwidth usage.

The Tensaurus architecture, which is explicitly shown in Figure 4, includes a 2D array of PEs, a tensor load unit (TLU), and a matrix load unit (MLU). It also incorporates scratchpad memories (SPMs) and a matrix store unit (MSU) for efficient data handling. The PEs could handle operations such as scalar-vector multiplication and element-wise vector operations. The TLU and MLU can manage the memory

accesses, while the MSU can accumulate partial results and store them into the output memory.

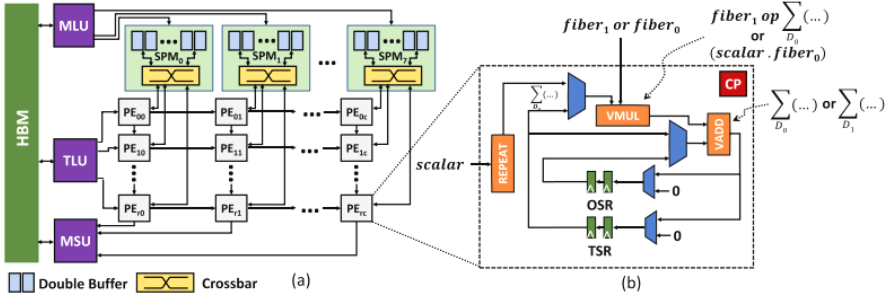


Fig. 4. The overall architecture of Tensaurus. MLU and TLU are used to manage the memory, while the MSU handles the computation and storage [9].

Tensaurus operates by reading sparse tensor data in CISS format, where non-zero elements are stored contiguously in memory for efficient access by the PEs. Each PE processes data in parallel, performing scalar-vector multiplications, element-wise vector multiplications, and accumulations in SIMD fashion. For sparse matrices, non-zero elements are processed, while dense matrices are managed by the system to avoid serialization in memory access. The design emphasizes load balancing and memory bandwidth utilization. Tensaurus delivers significant performance gains and energy savings over CPU/GPU baselines. For sparse tensor factorizations, it delivers $22.9\times$ speedup over CPUs and $3.1\times$ over GPUs. It also demonstrates $349.2\times$ speedup for CNN layers (AlexNet, VGG-16) compared to CPUs and $1.8\times$ faster than GPUs. Moreover, it is $1983.7\times$ more energy-efficient than CPUs and $226.6\times$ more efficient than GPUs for dense operations, making it a compelling choice for tensor computation tasks in machine learning.

3.4 Breakthroughs in Compute-in Memory Technique: GAS

Sparse matrix multiplication plays a crucial role in scientific computing, but traditional accelerators often suffer from performance limitations due to poor data locality and irregular memory access patterns. To address this memory bottleneck, in-memory computing (IMC) has emerged as a promising solution by enabling computation within memory. This paper presents a novel accelerator design that integrates IMC with sparse matrix multiplication, effectively overcoming the memory access constraints of conventional approaches. GAS employs a unified outer product-based multiplication methodology for the sparse matrix multiplication (SMM). It accelerates various types of multiplication, which includes SpMV, SpMSPv, SpMM, and SpMSPM. The accelerator uses content-addressable memory (CAM) for the index matching and multiply-add computation (MAC) arrays for in-situ floating-point computation. GAS integrates CAM arrays for index matching and MAC arrays for computational tasks. As is shown in Figure 5, this in-memory computing (IMC) architecture avoids the

traditional separation of memory and processing, thus optimizing performance and reducing memory access bottlenecks.

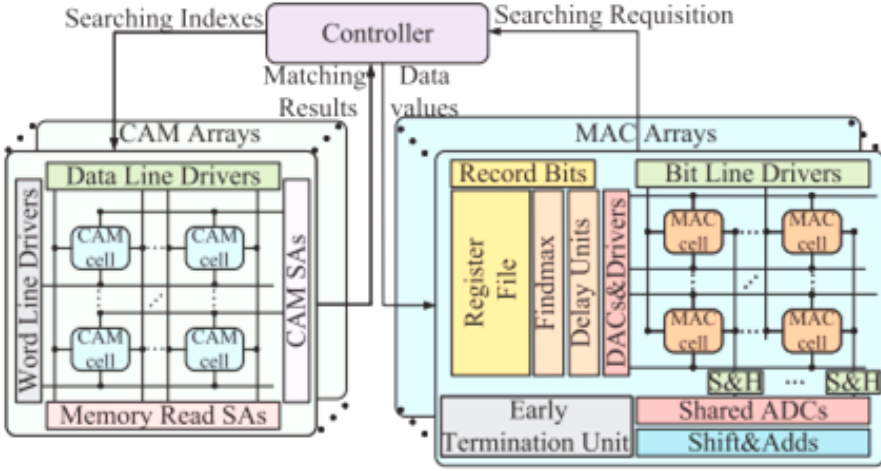


Fig. 5. The Integration of CAM, MAC and IMC. GAS uses CAM for indexing, MAC for computing and IMC for reducing the memory access [10].

The accelerator operates in three phases: Write, Load, and Compute. In the write phase, non-zero matrix entries are written to the CAM and MAC arrays. The load phase involves searching for the correct indices, followed by the compute phase, where in-situ multiplication occurs directly within the MAC arrays, improving energy efficiency. Experimental results show that GAS outperforms CPU and GPU implementations, achieving higher throughput and better energy efficiency in sparse matrix computations. It also outperforms other IMC-based accelerators, offering higher performance and energy efficiency while supporting sparser matrix multiplication functions. Its design flexibility and efficiency make it a promising solution for a wide range of computational tasks.

4 Conclusion

This work reviewed SOTA accelerators optimized for sparsity exploitation, efficient dataflows, and in-memory computing. Architectures like Eyeriss introduced row-stationary dataflows to reduce data movement and improve energy efficiency in CNNs. Cambricon-S, on the other hand, efficiently addressed sparse network irregularities through coarse-grained pruning and local quantization. Meanwhile, NullHop uses zero-skipping mechanism and sparse feature map compression method to maximize MAC utilization, and GAS employed a in-memory computing (IMC) approach to accelerate the computation in sparse matrix multiplication. Meanwhile, Tensaurus introduced a flexible accelerator capable of handling mixed tensor operations. By leveraging the CISS sparse storage format and optimized memory access patterns, Tensaurus achieved substantial improvements in both performance and energy efficiency. These works

collectively highlight the growing importance of sparsity-aware architectures, memory-efficient dataflows, and adaptive computation models in modern neural network accelerators. As deep learning workloads continue to expand in complexity and scale, future research should focus on balancing flexibility, efficiency, and computational power. Future research on hardware accelerators should emphasize innovative data reconfiguration techniques and advanced storage solutions to support diverse neural network models.

References

1. Park, S., K. Bong, D. Shin, J. Lee, S. Choi, and H.-J. Yoo, A 1.93TOPS/W scalable deep learning/inference processor with tetraparallel MIMD architecture for big-data applications, in Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC), Feb. 2015, pp. 1–3
2. Cavigelli, L., D. Gschwend, C. Mayer, S. Willi, B. Muheim, and L. Benini, Origami: A convolutional network accelerator, in Proc. 25th Ed. Great Lakes Symp. VLSI, 2015, pp. 199–204.
3. Chen, Y. et al.: DaDianNao: A machine-learning supercomputer, in Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture, 2014, pp. 609–622.
4. Zhang, S., Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, Cambricon-X: An Accelerator for Sparse Neural Networks, in 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 1-12
5. Han, S., X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, EIE: Efficient Inference Engine on Compressed Deep Neural Network, in 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), pp. 243-254
6. Chen, Y., T. Krishna, J. S. Emer, and V. Sze, Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks, in IEEE Journal of Solid-State Circuits **52**, 127–138 (2017)
7. Zhou, X., Z. Du, Q. Guo, S. Liu, C. Liu, C. Wang, X. Zhou, L. Li, T. Chen, and Y. Chen, “Cambricon-S: Addressing Irregularity in Sparse Neural Networks through A Cooperative Software/Hardware Approach,” in 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 1-15
8. Aimar, A., H. Mostafa, E. Calabrese, A. Rios-Navarro, R. Tapiador-Morales, I. Lungu, M. B. Milde, F. Corradi, A. Linares-Barranco, S. Liu, and T. Delbruck, NullHop: A Flexible Convolutional Neural Network Accelerator Based on Sparse Representations of Feature Maps, in IEEE Transactions on Neural Networks and Learning Systems **30**, 644–656 (2019)
9. Srivastava, N., H. Jin, S. Smith, H. Rong, D. Albonesi, and Z. Zhang, “Tensaurus: A Versatile Accelerator for Mixed Sparse-Dense Tensor Computations,” in 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 689-702
10. Zhang, X., Z. Li, R. Liu, X. Chen, and Y. Han, “GAS: General-Purpose In-Memory-Computing Accelerator for Sparse Matrix Multiplication,” in 2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA), pp. 230-243
11. Krizhevsky, A., I. Sutskever, and G. E. Hinton, ImageNet classification with deep convolutional neural networks, in Proc. Adv. Neural Inf. Process. Syst., **25**, 1097–1105(2012).
12. K. Simonyan and A. Zisserman, Very deep convolutional networks for large-scale image recognition, CoRR, abs/1409.1556, 1–14(2014).

13. Y. LeCun, K. Kavukcuoglu, and C. Farabet, Convolutional networks and applications in vision, in Proc. IEEE Int. Symp. Circuits Syst. (ISCAS), May/June. 2010, pp. 253–256.
14. He, K., X. Zhang, S. Ren, and J. Sun, Deep residual learning for image recognition, in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2016.
15. Zhang, C., P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks, in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Monterey California USA: ACM, Feb. 2015, pp. 161–170.
16. Ma, Y., Y. Cao, S. Vrudhula, and J. Seo, Optimizing Loop Operation and Dataflow in FPGA Acceleration of Deep Convolutional Neural Networks, in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Monterey California USA: ACM, Feb. 2017, pp. 45–54.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution-NonCommercial 4.0 International License (<http://creativecommons.org/licenses/by-nc/4.0/>), which permits any noncommercial use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

