2021 6th International Conference on UK-China Emerging Technologies

A Review of Convolutional Neural Networks Hardware Accelerators for AIoT Edge Computing

Fei Wu
University of Electronic Science and Technology of China
School of Information and Communication Engineering
Chengdu, China
wuf@std.uestc.edu.cn

Neng Zhao
University of Electronic Science and Technology of China
School of Information and Communication Engineering
Chengdu, China
zn@std.uestc.edu.cn

Ye Liu
University of Electronic Science and Technology of China
School of Information and Communication Engineering
Chengdu, China
liuye2018@std.uestc.edu.cn

Liang Chang
University of Electronic Science and Technology of China
School of Information and Communication Engineering
Chengdu, China
liangchang@uestc.edu.cn

Liang Zhou
University of Electronic Science and Technology of China
School of Information and Communication Engineering
Chengdu, China
zlzl@uestc.edu.cn

Jun Zhou
University of Electronic Science and Technology of China
School of Information and Communication Engineering
Chengdu, China
zhouj@uestc.edu.cn

2021 International Conference on UK-China Emerging Technologies (UCET) | 978-1-6654-9575-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/UCET54125.2021.9674972

***Abstract*—Convolutional neural networks (hereafter CNN) have been widely used in the Artificial Intelligence & Internet of Things (AIoT) applications, due to its powerful feature extraction and classification capabilities. However, with the scale of CNN models increasing, the large number of network parameters and operations pose a great challenge on the design of low latency and low power CNN hardware accelerator for AIoT edge computing. This paper reviews the design techniques of state-of-the-art CNN hardware accelerators, including the design methods of dense and sparse CNN hardware accelerators, optimization methods of CNN hardware accelerators through software and hardware co-design, and the advanced design methods of computing in memory. These methods may help understand how to design CNN hardware accelerators with high performance, low power, and small area by improving the utilization of processing elements (PEs), reducing invalid data movement and increasing hardware flexibility. Besides, this paper also discusses the future direction of the design of CNN hardware accelerators.**

Keywords*—convolutional neural networks, hardware accelerator, artificial intelligence & internet of things, edge computing

I. Introduction

The concept of artificial intelligence (AI) has been proposed since the last century, because of the development of the semiconductor industry, large amount of cheap data have been generated easily with the improvement of computing efficiency. It gives birth to many subversive neural network algorithms, comprising deep neural network (DNN), convolutional neural networks (CNN), recursive neural networks (RNN), long and short term memory (LSTM), and spiking neural network (SNN) etc. [1] Among them, CNN presents a powerful ability of feature extraction and classification which can be widely applied in internet of things (IoT) applications. CNN depends on the weight parameters sharing mechanism and its sparsely connected characteristic for artificial intelligence & internet of things (AIoT) edge computing including images and video object recognition and classification, natural language processing, indoor positioning, biomedical processing etc.

In recent years, with the development of CNN, an increasing number of CNN models have been designed. Although these models help in improving accuracy and the feature extraction ability, the number of parameters and the complexity of models are also increasing. It poses a great challenge on the design CNN hardware accelerators. Therefore, the academic world and industrial world both concentrate on exploring in-depth research on high performance and low energy cost CNN accelerators.

II. Background: Convolutonal Neural Networks

A. *Convolutional Neural Networks Algorithm*

CNN is a neural network based on the "receptive field" of biological neuroscience. It is evolved from multi-layer perceptron which generally consists of convolutional layer, activation layer, batch normalization layer, pooling layer and fully connected layer. Compared to other neural networks, CNN has the weight parameters sharing mechanism and sparsely connected characteristic:

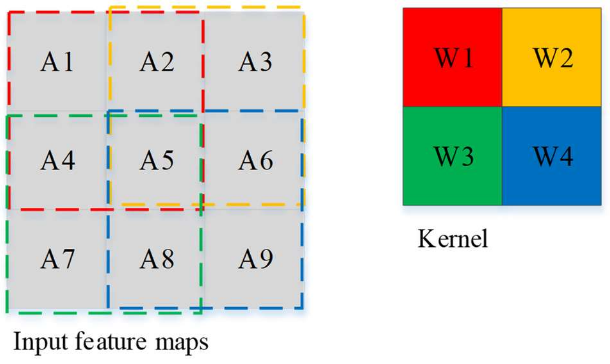

Fig. 1. Parameter sharing mechanism of CNN

1) *Weight Parameter Sharing*

Taking image processing as an example, the weights of the convolution kernel in the CNN slide on the feature map. As shown in Fig. 1, each weight in kernels is shared for multi-activation inputs. This mechanism reduces the amount of parameters in the convolution kernels.

2) Sparsely Connected

Different from traditional fully connected network as shown in the left side of Fig. 2, the connection of CNN is sparse. Each n-layer's neuron is connected with parts of n-1-layer's neurons and not all of the neurons as shown in the right side of Fig. 2. This characteristic decreases the complexity of CNN models.

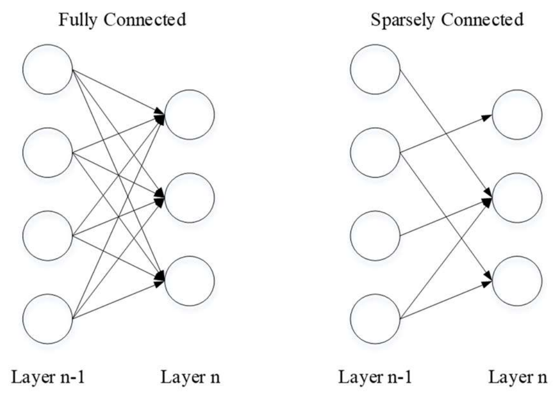

Fig. 2. Sparsely Connected characteristic of CNN

B. *The Basic Structure of Convolutional Neural Networks*

1) Convolution Layer

As shown in Fig. 3, the convolution (Conv) layer is composed of *M* input channels, the input feature maps (*Ifm*) with width *Wi* and height *Hi*, and *N* convolution kernels. Output feature maps (*Ofm*) are generated by the convolution operation of *Ifm* and kernels .The formula is shown in (1).

$$\boldsymbol{Ofm}[\mathrm{m}, h_o, w_o] = \sum_{m=0}^{M-1} \sum_{kh=0}^{KH-1} \sum_{kw=0}^{KW-1} \boldsymbol{K}[m,n,kh,kw] \times \boldsymbol{Ifm}[m, Sh_o + kh, Sw_o + kw] \qquad (1)$$

Among them, $0 \le n < N$, $0 \le m < M$, $0 \le h_o < H_o$, $0 \le w_o < W_o$, $0 \le kh < KH$, $0 \le kw < KW$, S is the step size of the convolution.

2) Activation Layer

The activation layer speeds up the learning process of feature maps by introducing nonlinear combination of features. There are many types of activation functions, such as sigmoid, tanh, maxout, ReLu, etc. ReLu is the most used activation function, The formula of ReLu is shown in (2).

$$\boldsymbol{f}(x) = \boldsymbol{max}(0, x) \qquad (2)$$

3) Batch Normalization Layer

The batch normalization layer can speed up the training process and avoid the saturation of the internal values. The formula is shown in (3).

$$\boldsymbol{y} = \frac{\gamma}{\sqrt{Va\ \ [x]+\epsilon}} \cdot \boldsymbol{x} + \left(\beta - \frac{\gamma E[x]}{\sqrt{Var[x]+\epsilon}}\right) \qquad (3)$$

Among them, *E[x]* is the mean value of the training neuron, *Var[x]*$+\epsilon$ is the standard deviation of the training neuron, γ and β are two learnable reconstruction parameters, make network learning recover the distribution of features to be learned.

4) Pooling Layer

The pooling layer can reduce the size of the feature maps, select the main features of the feature graph, and reduce the number of parameters. There are two common pooling methods: max pooling and average pooling.

5) Fully Connected Layer

As a classifier of the network, the fully connected layer maps the extracted features to the sample marker space. The formula is shown in (4).

$$\boldsymbol{Ofm}[m] = \sum_{m=0}^{M-1} \boldsymbol{K}[m,n]\boldsymbol{Ifm}[m] \qquad (4)$$

Among them, $0 \le n < N$, $0 \le m < M$,*N* is the number of output channels, and *M* is the number of input channels.

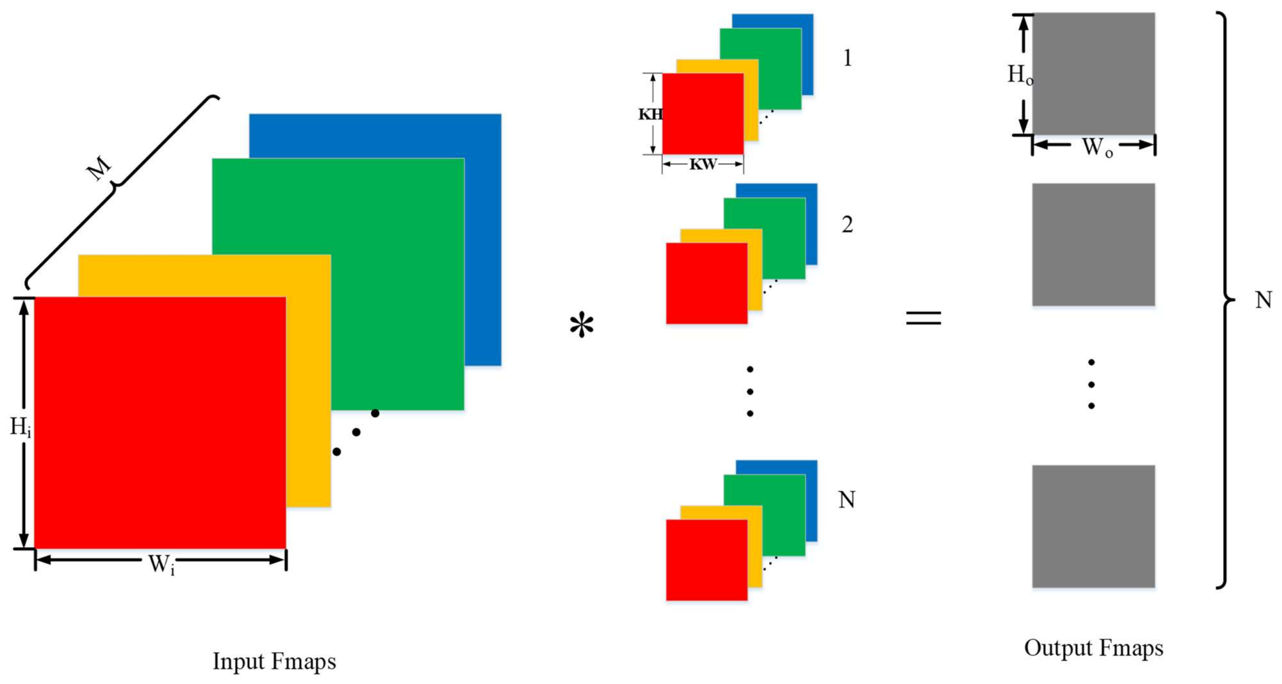

Fig. 3. The computation of CNN Conv layer

TABLE I. COMPARISON WITH DIFFERENT CNN MODULE

Model	Year	Depth	# Params	Input Size	ImageNet Top-5 Acc	New Features
LeNet [2]	1998	5	60K	32*32*1	-	The first CNNs
AlexNet [3]	2012	8	60M	224*224*3	79.1%	Dropout and ReLu
VGGNet-16 [4]	2014	16	138M	224*224*3	90.4%	Smaller structure
GoogleNet [5]	2015	22	4M	224*224*3	87.5%	Different kernel size
ResNet-50 [6] ResNet-152 [6]	2016	50 152	26M 60M	224*224*3	93.0% 93.8%	Residual learning
Inception-v3 [7] Inception-v4 [8]	2016 2017	48 77	24M 48M	299*299*3	94.4% 95.0%	Asymmetric convolutions Inception module and residual connection
DenseNet-169 [9]	2017	169	14M	224*224*3	93.2%	Skip connection
EfficientNet-B0 [10] EfficientNet-B7 [10]	2019	237 813	5.3M 66M	224*224*3	93.3% 97.0%	Combination of speed and precision

C. *Typical Convolutional Neural Networks Models*

Typical CNN models include LeNet [2], AlexNet [3], VGG [4], GoogleNet [5], ResNet [6], Inception [7,8], DenseNet [9], EfficientNet [10] as shown in Table I. With the improvement of CNN models, the parameters and depth of the models are more greater, the network models accuracy are more higher.

III. CONVOLUTIONAL NEURAL NETWORKS ACCELERATORS

A. *Spatial Architectures*

As the scale of CNN models increasing, the large number of network parameters and operations pose a great challenge on the design of low latency and low power CNN hardware accelerator for AIoT edge computing. AIoT edge computing needs high energy efficiency hardware, which can accelerate complex CNN algorithms with very low power consumption. The researchers explored high energy efficiency accelerator hardware design by building CNN's spatial architecture.

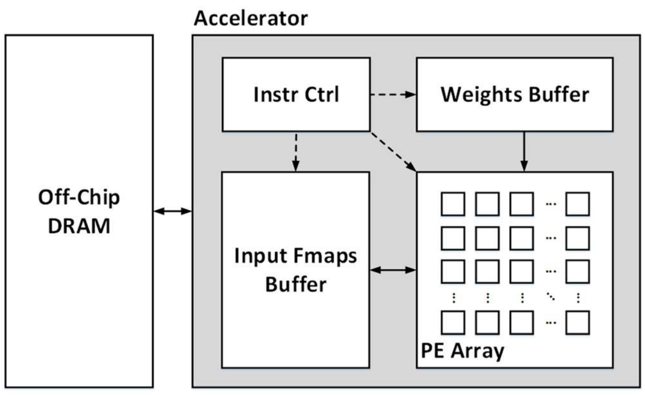

Fig. 4. Basic spatial architecture of CNN accelerator

The basic spatial architecture of the CNN accelerator is shown in Fig. 4, which consists of external DRAM and internal input feature maps buffer (IFB), weights buffer (WB), processing elements array (PEA) and instruction control unit (Instr Ctrl). The overall computing process is as follows: input feature maps and weights of external DRAM is moved into WB and IFB, and move the data of WB and IFB into PEA for computing under the control of Inst Ctrl. The computing sum is accumulated within PEA and the output is stored back to IFB. Then repeat the above process until the end of this layer computing.

CNN can be divided into dense and sparse types according to the complexity of network model. There are different optimization methods for these two types of network models.

1) *Dense Convolutional Neural Networks Accelerators*

Dense CNN refers to the original CNN model with unweighted compression or pruning operation. Such network model has large storage and computation, so it needs to be optimized from two aspects of computing and memory access:

a) Optimization of Computing

The simplest way to optimize CNN computing is to improve PE utilization and make each PE continuously compute through efficient pipeline. DianNao [11] proposed an accelerator that converts two dimensional convolution into two one-dimensional vector multiplication accumulative operations. This operation in ShiDianNao [12] and Eyeriss [13,14] became a two-dimensional PE array matching a higher dimensional convolution operation, with partial sum passing between the two more PEs.

TPU [15] also used a two-dimensional PE array, but it was different from the previous accelerator. TPU combined an efficient systolic computing method to form systolic array. It allowed the input feature maps flowing in one dimension, and it allowed the weights flowing in another dimension. When the data flows through the PE, the computing results are generated, and the partial sum is flushed to other PEs like a water flow. However, no matter what computing method is optimized, the purpose is to improve PE utilization and reduce computing latency.

b) Optimization of Memory Access

Because of the huge gap between external memory and internal memory access energy cost, an excellent memory access strategy can reduce the number of external memory accesses, and then reduce the energy cost of the accelerator. The method of minimizing access to external memory can be obtained by establishing DRAM access model and mapping method respectively [16,17]. The minimum communication bandwidth can also be obtained by establishing the memory access model of local memory [18-20]. The design and optimization of the memory access model can increase the reuse of data, reduce the invalid movement of data, and reduce the latency and energy cost of memory access.

Both computing and memory access are very important, and they need to be effectively combined. Only by making the memory access bandwidth and computing bandwidth as consistent as possible can we design energy efficient accelerator hardware.

2) Sparse Convolutional Neural Networks Accelerators

The research in [22,23] found that using pruning, compression and quantization methods to only learn important neural network connections can effectively reduce the amount of network parameters and the complexity of the model. And this method has no effect on the accuracy or only slight. This has also led to extensive research on the hardware design of this kind of sparse CNN accelerator.

a) Compression/Decompression of the Network Model

By compressing the network model, the storage and movement of parameters can be greatly reduced. Cnvlutin [24] used compressed sparse row (CSR) to compress the network model and implements it in hardware. Cambricon-X [25] used compressed Image Size (CIS) method to compress the input feature maps. EIE [26] used compressed sparse column (CSC) method to compress the weights. References [27,28] compressed both weights and input feature maps. Data are stored in memory by compression and decompressed during computing, which greatly reduces the storage and movement of data.

b) Zero-skipping Computing of Data

Sparse CNN has many zero-value inputs and weights. Skipping these invalid operations can reduce the power consumption of a large amount of data computing. References [25,29,30] indexed valid data addresses and skipped invalid zero-value data during storage and computing. Only processes valid multiplication and accumulation (MAC) operations. UCNN [31] reduced the invalid transportation of data by means of weights repetition and skipping computing of zero values.

Cambricon-S [32] dealt with the irregularity problem caused by the zero-skipping computing after the sparseness of the network. SqueezeFlow [33] used PT-OS sparse data stream to eliminate invalid operations while maintaining the regularity of CNN computing. Through the zero-skipping circuit design of the sparse CNN, only the non-zero values in the network are computed, which can reduce computational energy cost significantly.

B. *Software and Hardwawre co-design*

Different network models of CNN have different characteristics, so the hardware accelerator of CNN needs to have certain flexibility to adapt to variable network structures of different layers and different parameters under the same network, which requires the close co-design of software and hardware.

1) Design Programmable Instruction Set Architecture

By designing instruction set architecture (ISA), we can complete the mapping for different applications and different networks. Programmable instructions are adopted through ISA design to increase the reconfigurability of hardware. PuDianNao [34] and Cambricon [35] designed different instructions for their hardware accelerators. FlexACC [36] used RSIC-V instructions and an application-oriented ISA designed to enhance the coupling between instructions. The CNN accelerator generates programmable instructions through the compiler, and the hardware performs reconfigurable computation under the instructions decoding, so as to improve the flexibility of hardware.

2) Realize Data Flow Optimization

References [37-39] through the reconfigurable hardware analysis of a variety of data reuse models, found out the most effective collocation of data reuse models between different layers of the network. MPNA [40] not only supported a variety of convolution layer data reuse models, but also supported the acceleration of full connection layer to realize highly reconfigurable accelerator hardware.

3) Use Fixed-point Data and Variable Data Width

Through the quantitative method, the use of fixed-point hardware accelerator has only a slight impact on the accuracy of the algorithm, but the power consumption of data computing and movement in hardware is obviously reduced. Stripes [41] used fixed weights width and variable inputs width. UNPU [42] adopted fixed inputs width and variable weights width. References [43,44] used both variable weights and inputs width.

The comparison results of state-of-the-art CNN hardware accelerators are shown in Table II, including the comparison of platform, area, frequency, performance, power and energy efficiency.

TABLE II. COMPARISON WITH DIFFERENT CNN ACCELERATORS

	Platform (technology)	**Area (mm2)**	**Frequency (MHz)**	**Performance (GOPS)**	**Power (W)**	**Energy Efficiency (GOPS/W)**
DianNao[11]	ASIC (65nm)	3.02	980	452@16b	0.49	931.96
Eyeriss[13]	ASIC (65nm)	12.25	250	46@16b	0.28	166.20
TPU[15]	ASIC (28nm)	<331	700	92000@8b	40.00	2300.00
Accelerator1[21]	FPGA	-	150	137@16b	9.63	14.22
EIE[26]	ASIC (28nm)	40.80	800	102@4b	0.59	3000.00
NullHop[30]	ASIC (28nm) FPGA	6.30 -	500 60	471@16b 17@16b	0.16 2.30	1634.00 28.80
Cambricon-S[32]	ASIC (65nm)	6.73	1000	512@16b	0.80	641.16
DNA[37]	ASIC (65nm)	16.00	200	194@16b	0.48	405.85
FlexFlow[38]	ASIC (65nm)	3.89	100	420@16b	1.00	420.00
MPNA[40]	ASIC (28nm)	2.34	280	35.80@8b	0.24	149.70

C. *Computing in Memory*

Static random access memory (SRAM) or resistive random access memory (ReRAM) are used to computing in memory, which reduces the power consumption of data movement because the computing is directly computed in memory. Therefore, computing in memory is also the advanced of CNN hardware accelerator research. In order to solve the problem of "memory wall", references [45-47] proposed ReRAM based CNN accelerator. Part of ReRAM's crossbar array is configured as CNN application accelerator, and the microarchitecture and circuit design are realized, which can efficiently perform matrix vector multiplication. Reference [48] supported training and inference based on ReRAM and optimizes pipelines using inter-layer parallelism. This approach can greatly reduce the area overhead.

References [49,50] designed an in-memory CNN accelerator based on SRAM, which directly optimized MAC operations from the circuit level and increased data reuse of MAC operations at the convolution layer, thus improving energy efficiency and reducing area and energy overhead.

IV. Future of Convolutional Neural Netroks Hardware Accelerators

A. *Trade-off Performance and Flexibility*

At present, most designers of CNN accelerators focus on how to maximize accelerator performance. By designing a very dedicated accelerator to improve performance, this will greatly limit the flexibility of the accelerator. Inflexible hardware will be abandoned with the development of algorithms. Therefore, designers should make a trade-off in two aspects of high performance and flexibility.

B. *Training and Deployment*

Most of the accelerators are still aimed at the CNN inference, and the training of CNN is also an important process that cannot be ignored. How to improve the training efficiency of CNN is another key. In addition, the autonomous learning ability of the CNN accelerator needs to be considered. When an unlearned data enters the accelerator, the network needs to have certain autonomous learning ability, so as to avoid consuming too much time to retrain and redeploy the network model.

C. *Computing in Memory*

Computing in memory is undoubtedly the advanced of the current development of CNN accelerators. However, computing in memory is still carried out in the scope of academic research, and has not been widely applied in industry. Therefore, how to make the accelerators of memory computing applied in the industrial field as soon as possible is also a problem worth discussing.

D. *Software and Hardware Co-design*

The requirements of the application are changed at any time, even it may have different requirements in the same application. Therefore, the future of the accelerator must be highly co-design of software and hardware. It can achieve reconfigurable hardware in the shortest time through programmable software with reducing the cost of new requirements.

V. Conclusion

With the flourishing development of CNN in the AIoT applications, more and more CNN models have been proposed. The accuracy of the network model has been qualitatively improved, but it is accompanied by the complexity of the model and a large number of parameters, which poses a great challenge on the design of low latency and low power CNN hardware accelerator for AIoT edge computing. This paper reviews the design techniques of state-of-the-art CNN hardware accelerators, including the design methods of dense and sparse CNN hardware accelerators, optimization methods of CNN hardware accelerators through software and hardware co-design, and the advanced design methods of computing in memory. These methods may help understand how to design CNN hardware accelerators with high performance, low power, and small area by improving PE utilization, reducing invalid data movement and increasing hardware flexibility. Besides, this paper also discusses the future direction of the design of CNN hardware accelerators.

References

[1] Y. LeCun, Y. Bengio and G. Hinton, “Deep learning,” *Nature,* vol. 521, no. 7553, pp. 436-444, May. 2015.

[2] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, “Gradient-based learning applied to document recognition,” *in Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, Nov. 1998.

[3] A. Krizhevsky, I. Sutskever and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 25, no. 2, pp. 1097-1105, Jan. 2012.

[4] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *Computer Science*, arXiv preprint arXiv:1409.1556, Sep. 2014.

[5] C. Szegedy et al., “Going deeper with convolutions,” *in 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1-9.

[6] K. He, X. Zhang, S. Ren and J. Sun, “Deep Residual Learning for Image Recognition,” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770-778.

[7] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, “Rethinking the Inception Architecture for Computer Vision,” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2818-2826.

[8] C. Szegedy, S. Ioffe, V. Vanhoucke, “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning,” *In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, 2017, pp. 4278-4284.

[9] G. Huang, Z. Liu, L. V. D. Maaten and K. Q. Weinberger, “Densely Connected Convolutional Networks,” *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2261-2269.

[10] M. Tan and Q. V. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,” *Computer Science*, arXiv preprint arXiv:1905.11946, May. 2019.

[11] T. Chen et al., “DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning,” *International Conference on Architectural Support for Programming Languages & Operating Systems ACM*, Feb. 2014.

[12] Z. Du et al., “ShiDianNao: Shifting vision processing closer to the sensor,” *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015, pp. 92-104.

[13] Y. Chen, T. Krishna, J. Emer and V. Sze, “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks,” *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, 2016, pp. 262-263.

[14] Y. Chen, J. Emer and V. Sze, “Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks,” *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 367-379.

[15] N. P. Jouppi et al., “In-datacenter performance analysis of a tensor processing unit,” *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, pp. 1-12.

[16] J. Li et al., “SmartShuttle: Optimizing off-chip memory accesses for deep learning accelerators,” *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018, pp. 343-348.

[17] R. V. M. Pura, M. A. Hanif and M Shafique, “DRMap: A Generic DRAM Data Mapping Policy for Energy-Efficient Processing of Convolutional Neural Networks,” *Computer Science*, arXiv preprint arXiv:2004.10341, Apr. 2020.

[18] A. Stoutchinin, F. Conti and L. Benini, “Optimally Scheduling CNN Convolutions for Efficient Memory Access,” *Computer Science*, arXiv preprint arXiv:1902.01492, Feb. 2019.

[19] X. Chen, Y. Han and Y. Wang, “Communication Lower Bound in Convolution Accelerators,” *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 529-541.

[20] S. D. Manasi and S. S. Sapatnekar, “DeepOpt: Optimized Scheduling of CNN Workloads for ASIC-based Systolic Deep Learning Accelerators,” *ASPDAC '21: 26th Asia and South Pacific Design Automation Conference 2021*, 2021, pp. 235-241.

[21] J. Qiu et. al., “Going Deeper with Embedded FPGA Platform for Convolutional Neural Network,” *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2016, pp. 26-35.

[22] S. Srinivas and R. V. Babu, “Data-free Parameter Pruning for Deep Neural Networks,” *Computer Science*, arXiv preprint arXiv:1507.06149, Jul. 2015.

[23] S. Han, J. Pool, J. Tran and W. J. Dally, “Learning both Weights and Connections for Efficient Neural Networks,” *Proceedings of the 28th International Conference on Neural Information Processing Systems – Volume 1*, 2015, pp. 1135-1143.

[24] J. Albericio et al., “Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing,” *In Proceedings of the 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 1-13.

[25] S. Zhang et al., “Cambricon-X: An accelerator for sparse neural networks,” *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1-12.

[26] S. Han et al., “EIE: Efficient Inference Engine on Compressed Deep Neural Network,” *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 243-254.

[27] A. Parashar et al., “SCNN: An accelerator for compressed-sparse convolutional neural networks,” *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, pp. 27-40.

[28] E. Russo et al., “DNN Model Compression for IoT Domain Specific Hardware Accelerators,” *in IEEE Internet of Things Journal*, 2021, pp. 1-1.

[29] Y. Chen, T. Yang, J. Emer and V. Sze, “Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices,” *in IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292-308, Jun. 2019.

[30] A. Aimar et al., “NullHop: A Flexible Convolutional Neural Network Accelerator Based on Sparse Representations of Feature Maps,” i*n IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 3, pp. 644-656, Mar. 2019.

[31] K. Hegde et al., “UCNN: Exploiting Computational Reuse in Deep Neural Networks via Weight Repetition,” *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018, pp. 674-687.

[32] X. Zhou et al., “Cambricon-S: Addressing Irregularity in Sparse Neural Networks through A Cooperative Software/Hardware Approach,” *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018, pp. 15-28.

[33] J. Li et al., "SqueezeFlow: A Sparse CNN Accelerator Exploiting Concise Convolution Rules," *in IEEE Transactions on Computers*, vol. 68, no. 11, pp. 1663-1677, Nov. 2019.

[34] D. Liu et al. “PuDianNao: A Polyvalent Machine Learning Accelerator,” *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2015, pp. 369-81.

[35] S. Liu et al., “Cambricon: An Instruction Set Architecture for Neural Networks,” *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 393-405.

[36] E. -Y. Yang, T. Jia, D. Brooks and G. -Y. Wei, “FlexACC: A Programmable Accelerator with Application-Specific ISA for Flexible Deep Neural Network Inference,” *2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2021, pp. 266-273.

[37] F. Tu et al., “Deep Convolutional Neural Network Architecture With Reconfigurable Computation Patterns,” in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 25, no. 8, pp. 2220-2233, Aug. 2017.

[38] W. Lu et al., “FlexFlow: A Flexible Dataflow Accelerator Architecture for Convolutional Neural Networks,” *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 553-564.

[39] S. Yin et al., “A High Energy Efficient Reconfigurable Hybrid Neural Network Processor for Deep Learning Applications,” *in IEEE Journal of Solid-State Circuits*, vol. 53, no. 4, pp. 968-982, April 2018.

[40] M. A. Hanif et al., “MPNA: A Massively-Parallel Neural Array Accelerator with Dataflow Optimization for Convolutional Neural Networks,” *Computer Science*, arXiv preprint arXiv:1810.12910, Oct. 2018.

[41] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt and A. Moshovos, “Stripes: Bit-Serial Deep Neural Network Computing,” *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1-12.

[42] J. Lee, et al., “UNPU: An Energy-Efficient Deep Neural Network Accelerator With Fully Variable Weight Bit Precision,” *in IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, pp. 173-185. Oct. 2018.

[43] H. Sharma et al., “Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Networks,” *Computer Science*, arXiv preprint arXiv:1712.01507, Dec. 2017.

[44] S. Ryu, H. Kim, W. Yi and J. -J. Kim, “BitBlade: Area and Energy-Efficient Precision-Scalable Neural Network Accelerator with Bitwise Summation,” *In Proceedings of the 56th Annual Design Automation Conference 2019 (DAC '19),* 2019, pp. 1-6.

[45] P. Chi et al., “PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory,” *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 27-39.

[46] A. Shafiee et al., “ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars,” *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 14-26.

[47] H. Veluri, Y. Li, J. X. Niu, E. Zamburg and A. V. -Y. Thean, “High-Throughput, Area-Efficient, and Variation-Tolerant 3-D In-Memory Compute System for Deep Convolutional Neural Networks,” *in IEEE Internet of Things Journal*, vol. 8, no. 11, pp. 9219-9232. Feb. 2021.

[48] L. Song, X. Qian, H. Li and Y. Chen, “PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning,” *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 541-552.

[49] J. -W. Su et al., “16.3 A 28nm 384kb 6T-SRAM Computation-in-Memory Macro with 8b Precision for AI Edge Chips,” *2021 IEEE International Solid- State Circuits Conference (ISSCC),* 2021, pp. 250-252.

[50] Y. -D. Chih et al., “16.4 An 89TOPS/W and 16.3TOPS/mm2 All-Digital SRAM-Based Full-Precision Compute-In Memory Macro in 22nm for Machine-Learning Edge Applications,” *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, 2021, pp. 252-254.