2024 IEEE International Symposium on Circuits and Systems (ISCAS) | 979-8-3503-3099-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/ISCAS58744.2024.10558457

ACNNE: An Adaptive Convolution Engine for CNNs Acceleration Exploiting Partial Reconfiguration on FPGAs

Chun-Hsian Huang†
Dept. of Computer Science and Information Engineering
National Taitung University, Taiwan
†Email: huangch@nttu.edu.tw

Shao-Wei Tang and Pao-Ann Hsiung
Dept. of Computer Science and Information Engineering
National Chung Cheng University, Taiwan
Email: eric186699@gmail.com, pahsiung@ccu.edu.tw

***Abstract*—In this work, we propose a dynamical function exchange Convolutional Neural Networks (CNN) accelerator architecture named Adaptive CNN engine (ACNNE) that can reconfigure specific convolution layer hardware blocks according to model parameters at runtime. We mainly focus on exploiting reconfigurability for inferencing large-scale CNN on resource-constrained FPGAs. The proposed ACNNE can accelerate the process of convolution layers based on a nested-loop algorithm while a data buffering scheme is presented to reduce the iterations of memory accesses. As a study case, the VGG16 model was implemented on Xilinx ZU3EG MPSoC that can achieve 21.09 giga operations per second in the frequency of 100 MHz with a device resource utilization of 25% to 35%. Experiments show a single-image inference can be completed in 2.5 seconds with an average power consumption of 2.23 W, corresponding to a power efficiency of 9.46 GOPS/W.**

***Index Terms*—FPGA, convolutional neural network, partial reconfiguration, high-level synthesis.**

I. Introduction

Convolutional Neural Networks (CNNs) are commonly applied in numerous real-world applications. This kind of neural network is distinguished from other deep learning architectures by their superior performance with classification and computer vision tasks [1], [2]. Moreover, larger and deeper networks have been explored, and these models yield high accuracy that can presently surpass humans in image classification [3]. However, large-scale CNNs are computationally demanding and resource-consuming due to the convolutional operations and large number of parameters.

Recently, FPGAs have become a more appropriate alternative for CNNs accelerator as proposed recently [4]–[7]. Some implementations have also been presented to reduce the convolution layers latency [8], [9], partition the computations of the CNN model [10], [11] or integrate with FPGAs as prototyping platforms [12], [13]. Furthermore, FPGA-based accelerators have the advantages of power efficiency, high throughput, advanced reconfigurability, and fast development with high-level synthesis (HLS) [14].

Nowadays, many CNN models are usually implemented for supporting cloud services. However, embedded platforms are increasingly being used, and thus, there is a demand for integrating CNN applications with edge computing platforms, such as drones, mobile phones, or wearable devices. Therefore, an accelerator architecture that can infer a CNN model on a resource-constrained FPGA device without compromising the throughput or accuracy becomes a considerable choice. Furthermore, the reconfiguration capability of FPGAs also has general benefits for system architecture. Using the partial reconfiguration (PR) technique, the logic resources of each reconfigurable partition (RP) on an FPGA device can be dynamically reconfigured on demand.

There are two existing approaches using the PR technique to implement a larger-scale CNN on a resource-constrained FPGA device. An existing method is to reconfigure each convolution layer sequentially for performing all CNN layers [5]. Another is to divide a CNN model into multiple macroblocks, where each macroblock contains several convolution layers that can be performed by a pipeline strategy [15]. As a result, different CNN models can be adapted to the FPGA device, which also increases resource utilization significantly. However, this would not only incur the additional reconfiguration time overhead but also reduce system performance.

To solve the above issues, this work proposes an adaptive convolution engine (ACNNE) based on partial layers of a CNN working concurrently in a pipelined style. The contributions of this work are listed as follows.

- Different CNN architectures can be reconfigured based on the size of kernel filters in CNN layers without affecting the other system components.
- A nested-loop algorithm and a data buffering scheme are presented to reduce the iterations of memory accesses for the operations of convolution layers.
- Hardware intellectual properties (IPs) for convolution layers are deployed in C++ and synthesized with HLS to ease the block design burden for various CNNs.

The rest of this paper is organized as follows. Section II describes the related work and Section III introduces our proposed ACNNE design. Section IV describes our experiments and analyses, while Section V concludes this work.

Authorized licensed use limited to: PES University Bengaluru. Downloaded on May 16,2025 at 10:56:00 UTC from IEEE Xplore. Restrictions apply.

II. Related work

For FPGA-based CNN accelerators, there are three categories, including the streaming architectures [16]–[19], the single computation engines [20]–[23] and reconfigurable designs [5], [24], [25].

The most existing streaming architectures [16]–[19] can achieve good performance or high accuracy. Venieris et al. [16] proposed a framework fpgaConvNet mapping CNNs to particular FPGA platforms. They exploited the synchronous dataflow (SDF) model to capture CNN workloads as streaming computations. Baskin et al. [18] also presented a streaming model based on the functional decomposition of the calculations. The input data size could extend to 144×144, while the inference could be performed on resource-constrained FPGA devices. An end-to-end framework, namely FINN-R [17], was presented for fast exploration of Quantized Neural Networks (QNNs). FINN-R consisted of a cascade multilayer IP for pipelined architectures. Due to the quantization of weights and activations, the required size of memory would be decreased so that the parameters of weights and activations could be easily stored in the on-chip memory of an FPGA device. Alemdar et al. [19] proposed a ternary-weight neural network (TNN) with streaming architecture. TNN achieved 255K frames per second on the MNIST dataset.

The designs based on a single computation engine [20]–[23] could achieve good scalability or resource utility. A systolic array-like architecture [20] used a theoretical roofline model to build accelerators optimized for each layer's execution. Ovtcharov et al. [21] proposed an architecture to accelerate cloud computing of CNNs server solution with multiple FPGA cards. The architecture used a top-level controller to control data flow with a PCI memory interface. Judd et al. [22] also built a bit-serial processor capable of processing multiple precisions on a single calculated array, while Shuang et al. [23] proposed an FP-BNN model implemented on a single processing engine for BNN using XNOR pop-count data path. Batch normalization was also used, and the weight data were scaled to a floating point.

To enhance design flexibility, recently, the partial reconfiguration (PR) technique [26] has also been applied to the CNN design. Youssef et al. [24] presented a flexible CNN accelerator that exploits PR to adjust power consumption depending on the battery level. Meloni et al. [25] developed an accelerator architecture integrated with a highly parallel Hardware Convolution Engine (HWCE) based on a tightly coupled computing cluster. The proposed accelerator could be reconfigured at runtime for performing convolutions for different filter sizes in different strides. Furthermore, Florian et al. [5] exploited the PR technique to reconfigure the processing engines that operate in a cascaded manner. Based on the above discussion, in this work, we explore the PR capability to enhance system performance by proposing a pipelined architecture. Furthermore, a data reuse scheme, along with the use of the block ram (BRAM) on an FPGA, is presented to reduce the iterations of data access for an off-chip dynamic random access memory (DRAM). Details are given in Section III.

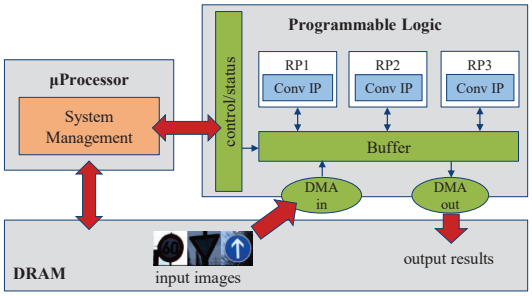


Fig. 1. ACNNE design

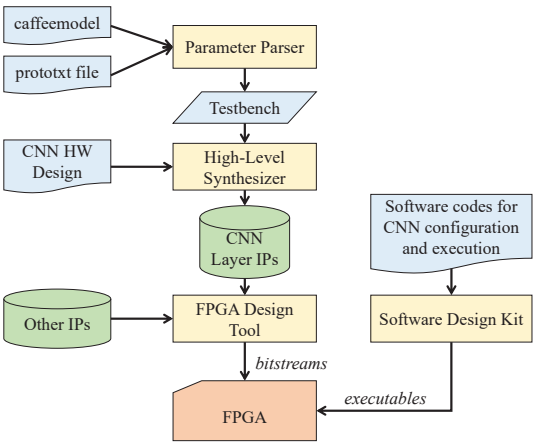


Fig. 2. Development flow

III. Proposed ACNNE design

To support not only the acceleration of convolution layers but also system adaptivity, the ACNNE design is thus proposed, as shown in Figure 1. In the programmable logic, some reconfigurable partitions (RPs) are designed to configure the hardware IPs for convolution layers of a CNN model. A buffer is used to store and transfer data between feature maps, while the direct memory access (DMA) controller is responsible for the data transmission between the DRAM and the buffer. A system management mechanism executed on the microprocessor is responsible for configuring the RPs and controlling the operations of the CNN model.

A. Acceleration of Convolution Layers

To accelerate the computations of the convolution layers efficiently, in this work, we propose a design method based on the hardware implementation as shown in Algorithm 1. Here, all data of a single convolution layer are processed from the upper-left pixel to the bottom-right one. Figure 2 shows our design development flow. Firstly, the network description of a CNN model is extracted by a parameter parser whose input is a caffe prototxt file and a caffe pre-trained model file, including binary weight and bias. Then, the generated file that contains the parameters of each layer of a CNN model

Authorized licensed use limited to: PES University Bengaluru. Downloaded on May 16,2025 at 10:56:00 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1: Acceleration of Convolution Layers

Init:
Buf_{in}, Buf_{out}: The buffers to save the pixel values of the input feature map and output one;
Buf_{para}: The buffer to save the weights of the CNN;
I_{map}: The input feature map;
$parameters$: The trained network parameters;
H_{out}, W_{out}: The height and the weight of the output feature map;
FW_{size}: The size of the filter window; $Depth_{in}$: The depth of the input feature map;
$Depth_{out}$: The depth of the output feature map;
$pixel_{in}$: The value of a specific input pixel;
$pixel_{out}$: The value of a specific output pixel;
$filter$: The value of a specific filter element on the filter window;

1 **Function** `ConvHW(`I_{map}, H_{out}, W_{out}, FW_{szie}, $Depth_{in}$, $Depth_{out}$, $parameters$`)`**:**
2 $Buf_{in} \leftarrow I_{map}$;
3 $Buf_{para} \leftarrow parameters$;
4 $Buf_{out} \leftarrow 0$;
5 **for** $w = 0$ *to* $H_{out} - 1$ **do**
6 **for** $x = 0$ *to* $W_{out} - 1$ **do**
7 **for** $y = 0$ *to* $Depth_{in} - 1$ **do**
8 **for** $z = 0$ *to* $Depth_{out} - 1$ **do**
9 **for** $i = 0$ *to* $FW_{szie} - 1$ **do**
`// For the weight of the filter window;`
10 **for** $j = 0$ *to* $FW_{szie} - 1$ **do**
`// For the height of the filter window;`
11 $filter = Buf_{para}(y, z, i, j)$;
12 $pixel_{in} = Buf_{in}(w + i, x + j, y)$;
13 $pixel_{out} = pixel_{out} + filter \cdot pixel_{in}$;
14 $Buf_{out}(w, x, z) = Buf_{out}(w, x, z) + pixel_{out}$;
15 **for** $l = 0$ *to* $Depth_{out} - 1$ **do**
16 $DRAM(w, x, l) = Buf_{out}(w, x, l)$
`// Add Bias and applying ReLU in the software side;`

is used for the functional validation (testbench) via a high-level synthesizer (HLS). Based on the CNN design, by using the HLS, the layer-specific hardware designs are generated and packed as hardware IPs. Based on the CNN layer IPs and other system IPs, the corresponding bitstreams using the FPGA design tool are generated, while the software executables for the CNN configuration and execution are generated.

Based on Algorithm 1, we take advantage of the independence of intra-kernel multiplications to enhance the parallelization to satisfy the prerequisites of the pipeline architecture. Through the support of HLS, the loop operations used for the output feature map (Line 8) are fully unrolled so that two continuous iterations can be performed directly. The loop operations used for the filter window (Lines 9 and 10) are also unrolled to reduce the time of the entry and exit for each iteration. However, the output buffer needs to interact with the DRAM frequently, which thus becomes a performance bottleneck of the pipeline. Here, the resources for latency are used in the output buffer to save the convolution results temporally (Lines 15 and 16).

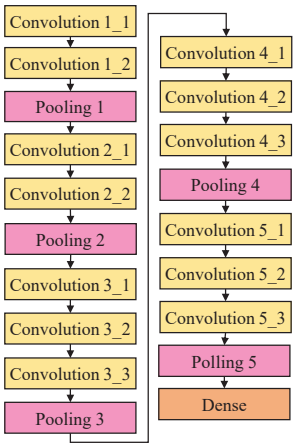


Fig. 3. Illustration example - VGG16

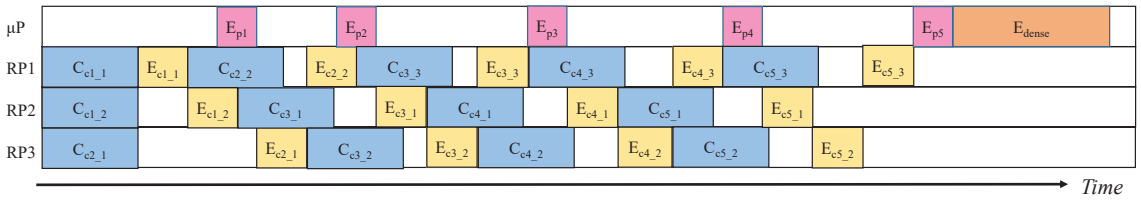
According to the above steps using HLS, when the size of the filter window is FW_{size}, FW_{size}^2 multiplications can be performed based on an adder tree. This indicates the operations can be processed in parallel when the required data are available. Furthermore, to reduce the iterations of DRAM access, three types of buffers, including the input buffer (Buf_{in}), the output one (Buf_{out}), and the parameter one (Buf_{para}), are also implemented by using the BRAM blocks in the FPGA. An input buffer for each pixel of the input feature map is designed, where the values of each pixel are saved for data reuse until the last filter is performed. An output buffer is designed to save each pixel value of the output feature map, while the data are transferred to the DRAM in the burst mode through the assistance of the DMA controller. Further, a parameter buffer is used to save all the weight values of the convolution layer, and the corresponding size is $FW_{size} \times FW_{size} \times Depth_{in} \times Depth_{out}$. To reduce the usage of the on-chip memory usage the quantization process is applied to the parameters of a CNN model to reduce the use of the on-chip memory usage.

B. System management

Based on the acceleration of convolution layers as described in Section III-A, convolution layers can be implemented as the corresponding hardware designs. By further realizing these hardware designs as reconfigurable hardware modules through the PR technique [26], a large-scale CNN model can be performed even though the total available amount of logic resources is less than that required by a CNN model.

Take a CNN model called VGG16 [27] as our illustration example. As shown in Figures 3 and 4, based on the proposed ACNNE design containing three RPs, as shown in Figure 1, three RPs can be dynamically configured as the

Authorized licensed use limited to: PES University Bengaluru. Downloaded on May 16,2025 at 10:56:00 UTC from IEEE Xplore. Restrictions apply.



C_{ci}: Configure the convolution layer i; E_{ci}: Execute the convolution layer i; E_{pi}: Execute the pooling layer i; E_{dense}: Execute the dense layer i;

Fig. 4. Execution results

TABLE I
THREE SYSTEM DESIGNS

Imp.	Resource utilization				Config. time	Infer. time
	LUT	FF	BRAM	DSP		
1RP_1	32.7%	28.5%	31.6%	21.1%	114 ms	2,741.1 ms
2RP_1	31.9%	28.5%	31.6%	21.1%	114 ms	2,582.1 ms
2RP_2	32.7%	28.1%	29.4%	21.1%	105 ms	
3RP_1	32.7%	28.1%	29.4%	21.1%	114 ms	
3RP_2	28.2%	27.5%	28.9%	14.7%	97 ms	2,435.1 ms
3RP_3	31.9%	28.5%	31.6%	21.1%	105 ms	

iRP_j: The jth RP in the system design having i RP(s). Imp.: System implementation. FF: Flip-flop. Bit. size: Bitstream size. Config. time: Configuration time.

requested convolution layers. Furthermore, using the configuration prefetch [28], the convolution layers are configured in `RP1`, `RP2`, and `RP3` in advance instead of configuring them when requesting to be executed. As a result, through the configuration and execution of three RPs and the incorporation with the microprocessor, the VGG16 model can be performed.

IV. Experiments

To demonstrate the practicability of our proposed method, the AVNET Ultra96-V2 platform with an AMD Xilinx Zynq UltraScale+MPSoC ZU3EG A484 FPGA device was used to implement the ACNNE design, while the VGG16 model trained on the ImageNet dataset [33] with caffe framework was adopted as the CNN model used. Ristretto [34] was used to convert the weight and bias of the Caffe model from floating-point to fixed-point. Furthermore, the hardware modules for some specific convolution layers were implemented as individual partial bitstreams, while the processor configuration access port (PCAP) controller was used to configure these partial bitstreams.

Three system designs individually having one RP, two RPs, and three RPs were implemented for comparison. The experimental results for resource utilization, configuration time, and inference time are given in Table I. We can observe that more RPs can reduce the idle time incurred by waiting for reconfiguration, and thus, the inference time of the VGG16 model can be accordingly reduced.

To further compare the proposed ACNNE design with the state-of-the-art work, more indicators, such as performance, power efficiency, and density performance, are used in the experiment. The results are given in Table II. In most existing work, only convolutional layers were used to evaluate the corresponding power efficiency and performance. Therefore, in this experiment, the power efficiency and performance of convolution layers in the ACNNE were also evaluated. According to the experiment, we can observe that ACCNE provides lower performance than the comparable state-of-the-art solutions do. However, the evaluations of performance density are better than most other solutions except Caffeine [31]. Furthermore, the amount of logic resources in the FPGA device (ZU3EG) used in this work is less than or equal to those used in the comparable state-of-the-art solutions. Due to the adaptability, CNN models with bigger network sizes than VGG16 can also be performed in the ACNNE design. This means that when the number of logic resources in the FPGA device is limited and the performance requirement can be satisfied, ACNNE can perform more different CNN models. Therefore, ACNNE is very attractive to resource-constrained edge AI platforms.

V. Conclusions

This work proposes an FPGA-based CNN accelerator named ACNNE that can perform different CNN models even though the total available amount of logic resources is less than that required by a CNN model. Compared to state-of-the-art solutions, ACNNE has the advantage of low resource usage and scalability.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, vol. 25, Dec. 2012.

[2] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *Advances in Neural Information Processing Systems*, vol. 28, December 2015.

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

[4] E. Nurvitadhi, J. Sim, D. Sheffield, A. Mishra, S. Krishnan, and D. Marr, "Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC," in *Proc. Int. Conf. Field Programmable Logic and Applications*. IEEE, Aug. 2016, pp. 1–4.

[5] F. Kästner, B. Janßen, F. Kautz, M. Hübner, and G. Corradi, "Hardware/software codesign for convolutional neural networks exploiting dynamic partial reconfiguration on pynq," in *Proc. IEEE Int. Parallel and Distributed Processing Symposium Workshops*. IEEE, May 2018, pp. 154–161.

[6] M. Farhadi, M. Ghasemi, and Y. Yang, "A novel design of adaptive and hierarchical convolutional neural networks using partial reconfiguration on FPGA," in *Proc. IEEE High Performance Extreme Computing Conf.* IEEE, Sep. 2019, pp. 1–7.

Authorized licensed use limited to: PES University Bengaluru. Downloaded on May 16,2025 at 10:56:00 UTC from IEEE Xplore. Restrictions apply.

TABLE II
COMPARISON WITH STATE-OF-THE-ART IMPLEMENTATION, INCLUDED FPGACONVNET [16], ANGEL-EYE [29], DNNWEAVER [30], CAFFEINE [31], CNNA [32]

Arch.	Power Efficiency [GOPS/W]	Density [GOPS/DSP]	Density [GOPS/kLUT]	Perfor. [GOPS]	Device	Freq. [MHz]	Fixed bits
fpgaConvNet	7.27	0.221	0.91	48.5	ZC7Z020	125	16
fpgaConvNet	-	0.173	0.71	156.0	ZC7Z045	125	16
Angel-Eye	24.10	-	-	85.3	ZC7Z020	214	8
Angel-Eye	-	0.860	0.21	188.0	ZC7Z045	150	16
DnnWeaver	-	0.143	0.59	31.4	ZC7Z020	150	16
Caffeine	12.40	0.187	1.55	310.0	KU060	200	16
$CNNA_{16}$	11.86	0.078	0.62	26.4	ZU3EG	100	16
$CNNA_8$	22.94	0.110	0.61	38.0	ZU3EG	172	8
ACNNE (Ours)	9.46	0.277	0.91	21.1	ZU3EG	100	8

GPOS: Giga operations per second.

[7] B. Seyoum, M. Pagani, A. Biondi, S. Balleri, and G. Buttazzo, "Spatio-temporal optimization of deep neural networks for reconfigurable FPGA SoCs," *IEEE Trans. Computers*, vol. 70, no. 11, pp. 1988–2000, Dec. 2020.

[8] D. Wu, Y. Zhang, X. Jia, L. Tian, T. Li, L. Sui, D. Xie, and Y. Shan, "A high-performance CNN processor based on FPGA for mobilenets," in *Proc. Int. Conf. Field Programmable Logic and Applications*. IEEE, Nov. 2019, pp. 136–143.

[9] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2016, pp. 779–788.

[10] Y. Shen, M. Ferdman, and P. Milder, "Maximizing CNN accelerator efficiency through resource partitioning," in *Proc. ACM/IEEE 44th Annual Int. Symposium on Computer Architecture*. IEEE, Jan. 2017, pp. 535–547.

[11] W. Zhang, J. Zhang, M. Shen, G. Luo, and N. Xiao, "An efficient mapping approach to large-scale dnns on multi-FPGA architectures," in *Proc. Design, Automation & Test in Europe Conf. & Exhibition*. IEEE, May 2019, pp. 1241–1244.

[12] X. Xu and B. Liu, "FCLNN: A flexible framework for fast CNN prototyping on FPGA with opencl and caffe," in *Proc. Int. Conf. Field-Programmable Technology*. IEEE, Dec. 2018, pp. 238–241.

[13] L. Wang, Y. Zhao, and X. Li, "An automatic conversion tool for caffe neural network configuration oriented to opencl-based FPGA platforms," in *Proc. IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conf.* IEEE, Mar. 2019, pp. 195–198.

[14] D. Baptista, F. Morgado-Dias, and L. Sousa, "A platform based on hls to implement a generic CNN on an FPGA," in *Proc. Int. Conf. Engineering Applications*. IEEE, Oct. 2019, pp. 1–7.

[15] H. Irmak, D. Ziener, and N. Alachiotis, "Increasing flexibility of FPGA-based CNN accelerators with dynamic partial reconfiguration," in *Proc. Int. Conf. Field-Programmable Logic and Applications*. IEEE, Aug. 2021, pp. 306–311.

[16] S. I. Venieris and C.-S. Bouganis, "FPGAConvNet: A framework for mapping convolutional neural networks on FPGAs," in *Proc. IEEE 24th Annual Int. Symposium on Field-Programmable Custom Computing Machines*. IEEE, Aug. 2016, pp. 40–47.

[17] M. Blott, T. B. Preußer, N. J. Fraser, G. Gambardella, K. O'brien, Y. Umuroglu, M. Leeser, and K. Vissers, "FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks," *ACM Trans. Reconfigurable Technology and Systems*, vol. 11, no. 3, pp. 1–23, Mar. 2018.

[18] A. Prost-Boucle, A. Bourge, F. Pétrot, H. Alemdar, N. Caldwell, and V. Leroy, "Scalable high-performance architecture for convolutional ternary neural networks on FPGA," in *Proc. 27th Int. Conf. Field Programmable Logic and Applications*. IEEE, Sep. 2017, pp. 1–7.

[19] H. Alemdar, V. Leroy, A. Prost-Boucle, and F. Pétrot, "Ternary neural networks for resource-efficient ai applications," in *Proc. Int. Joint Conference on Neural Networks*. IEEE, May 2017, pp. 2547–2554.

[20] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. 2015 ACM/SIGDA Int. Symposium on Field-programmable Gate Arrays*, Feb. 2015, pp. 161–170.

[21] K. Ovtcharov, O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss, and E. S. Chung, "Accelerating deep convolutional neural networks using specialized hardware," *Microsoft Research Whitepaper*, vol. 2, no. 11, pp. 1–4, Feb. 2015.

[22] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in *Proc. Annual IEEE/ACM 49th Int. Symposium on Microarchitecture*. IEEE, Dec. 2016, pp. 1–12.

[23] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, "FP-BNN: Binarized neural network on FPGA," *Neurocomputing*, vol. 275, pp. 1072–1086, Jan. 2018.

[24] E. Youssef, H. A. Elsemary, M. A. El-Moursy, A. Khattab, and H. Mostafa, "Energy adaptive convolution neural network using dynamic partial reconfiguration," in *Proc. IEEE 63rd Int. Midwest Symposium on Circuits and Systems*. IEEE, Sep. 2020, pp. 325–328.

[25] P. Meloni, G. Deriu, F. Conti, I. Loi, L. Raffo, and L. Benini, "A high-efficiency runtime reconfigurable ip for CNN acceleration on a mid-range all-programmable soc," in *Proc. Int. Conf. ReConFigurable Computing and FPGAs*. IEEE, Nov. 2016, pp. 1–8.

[26] AMD-Xilinx, "Vivado design suite user guide - dynamic function exchange UG909(v2020.2)," 2022.

[27] Z. Liu, J. Wu, L. Fu, Y. Majeed, Y. Feng, R. Li, and Y. Cui, "Improved kiwifruit detection using pre-trained VGG16 with RGB and NIR information fusion," *IEEE Access*, vol. 8, pp. 2327–2336, 2020.

[28] S. Banerjee, E. Bozorgzadeh, and N. Dutt, "Physically-aware HW-SW Partitioning for Reconfigurable Architectures With Partial Dynamic Reconfiguration," in *Proc. 42nd ACM/IEEE Design Automation Conf.*, Jun. 2005, pp. 335–340.

[29] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping CNN onto embedded FPGA," *IEEE Trans. Computer-aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 35–47, May 2017.

[30] H. Sharma, J. Park, E. Amaro, B. Thwaites, P. Kotha, A. Gupta, J. K. Kim, A. Mishra, and H. Esmaeilzadeh, "DNNweaver: From high-level deep network models to FPGA acceleration," in *Proc. Workshop on Cognitive Architectures*, Dec. 2016.

[31] C. Zhang, G. Sun, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 11, pp. 2072–2085, Jan. 2018.

[32] K. Bjerge, J. H. Schougaard, and D. E. Larsen, "A scalable and efficient convolutional neural network accelerator using HLS for a system-on-chip design," *Microprocessors and Microsystems*, vol. 87, p. 104363, Nov. 2021.

[33] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*. Ieee, Jun. 2009, pp. 248–255.

[34] P. Gysel, J. Pimentel, M. Motamedi, and S. Ghiasi, "Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks," *IEEE Trans. Neural Networks and Learning Systems*, vol. 29, no. 11, pp. 5784–5789, Mar. 2018.

Authorized licensed use limited to: PES University Bengaluru. Downloaded on May 16,2025 at 10:56:00 UTC from IEEE Xplore. Restrictions apply.