A Winograd-Convolution-Based Accelerator on FPGA for Real-time Object Detection with Effective On-chip Buffer Access Patterns

Keehyuk Lee*, Mincheol Cha*, Soosung Kim†, Xuan Truong Nguyen*, Hyuk-Jae Lee*
*Department of Electrical and Computer Engineering, Seoul National University, South Korea
†School of Electical and Electronics Engineering, Chung-Ang University, South Korea
{lkh099, mccha, truongnx, hjlee}@capp.snu.ac.kr, kimsoosung@cau.ac.kr

***Abstract*—A convolutional neural network (CNN) based object detector typically requires billions of multiplications, making it difficult to achieve a real-time end-to-end inference on FPGA. To reduce the number of multiplications, Winograd convolution (WinoConv) appears promising in accelerating CNN-based object detectors. However, designing a generic real-time WinoConv-based accelerator on FPGA is challenging since it requires effectively executing various layers, for example, 3×3 and 1×1 layers with strides 1 and 2. To address the problem, this paper proposes a WinoConv-based engine on FPGA for real-time object detection with effective line buffer access patterns. Specifically, we propose a simple yet effective dataflow to process tiled input and output tensors with a stride of 2, mitigating an overhead to reorder feature maps between layers. To enable practical use of the engine, we introduce a hardware-friendly post-training quantization scheme, achieving a negligible accuracy drop, i.e., less than 1% in mean average precision (mAP) for a Tiny-YOLOv3 model on the VOC dataset. Our engine is implemented with Verilog HDL and deployed on a Xilinx Zynq ZCU106 FPGA board. The experimental results show that our engine achieves a latency of 5.134ms to inference a custom Tiny-YOLOv3 model, resulting in a throughput of 292.18 GOPS at the clock frequency of 125 MHz on a real-time video stream. The engine utilizes 1053 DSPs and 105 BRAM.**

***Index Terms*—Winograd Convolution, Data Reordering, Real-Time, FPGA, Accelerator, YOLOV3-tiny**

I. Introduction

Over the past decade, Convolutional Neural Networks (CNNs) have demonstrated impressive performance in image processing tasks, such as object detection and image segmentation [1]. However, the large computational complexity of CNN models incurs extensive computation time and power, while the high space complexity for storing model weights requires frequent DDR access [2]. Numerous studies have stated the advantages of inference on FPGA over CPU and GPU, such as employing custom datapath to minimize DDR access and power while maximizing parallelism to reduce latency [4]–[7]. Many works have focused on deploying single-stage object detectors such as YOLO (You Only Look Once) model for fast and efficient multiple object detection [9]–[13].

To address the model's computational demands, various techniques such as layer fusion, quantization, and Winograd convolution have been adopted [14]–[16]. INT8 quantization can reduce resource usage and speed up computation on FPGAs, in exchange for an accuracy drop of 1% or higher [16], [18]. WinoConv is a promising technique for optimizing FPGA deployments, as it reduces computational complexity and enhances resource utilization, throughput, and efficiency [15]. However, since WinoConv doubles stride to 2, additional data reshaping is necessary, increasing buffer space and latency overhead. Zhang et al. described a Jump-step flow algorithm that utilizes on-chip buffers to pipeline reshaped data [17]. However, as reshaping relied solely on dynamic buffer access, hardware and latency overhead could not be circumvented.

To tackle these challenges, we propose an architecture featuring a reordered data format optimized for WinoConv and a flexible INT8 quantization method to maximize FPGA resource utilization. Our key contributions include:

- Introducing a novel data reordering and scheduling scheme for stride 2, which significantly reduces data buffering and reordering overhead.
- Enhancing the INT8 quantization method to limit the mAP reduction to less than 1%.

Running YOLOv3-tiny, our accelerator achieves real-time inference with a latency of 5.134ms, delivering 292.18 GOPS at 125 MHz. When implemented on a Xilinx Zynq+ ZCU106 board, it utilizes 1053 DSPs and 105 BRAM.

II. Background

A. YoloV3Tiny Specifications

For lightweight acceleration of object detection, we chose the CNN model YoloV3Tiny [19]. The model consists of 6 types of layers : convolution (CONV), batch normalization (BN), leaky ReLU, max pooling (MaxPool), concatenation (Concat), and upsampling (Upsample). The custom YOLOv3-tiny model and dataset used in our work is tailored for multi-camera object detection in unmanned stores [18], [23]. The number of filters, layers and input image size have been decreased, reducing computation by 73.3%. Consequently, both inference time and buffer size are significantly reduced.

B. Quantization

Symmetric and asymmetric quantization are prevalent techniques in neural network quantization. Symmetric quantization maps floating-point values to a range symmetric around zero, simplifying hardware implementation by using a uniform scale factor for positive and negative values. While efficient, it

may underutilize the representation range if data is not zero-centered, potentially reducing precision. Conversely, asymmetric quantization uses distinct scales and zero points, offering greater flexibility and precision, especially for skewed data distributions, though at the expense of increased hardware complexity.

In the YOLO network, which includes CONV, BN, and activation layers, quantization leads to precision loss, particularly due to the BN layer's running mean and variance. This loss propagates through the network, reducing the overall model accuracy cumulatively with each quantization step.

C. Winograd Convolution Algorithm

The Winograd convolution algorithm aims to decrease the number of multiplications, a key bottleneck in CNN accelerators. The two-dimensional Winograd algorithm F(2x2, 3x3) takes in a 4x4 IFM window **d** and 3x3 weight **g** as input and produces a 2x2 output feature map(OFM) window **Y**. The diagram for WinoConv is shown in Fig. 1.

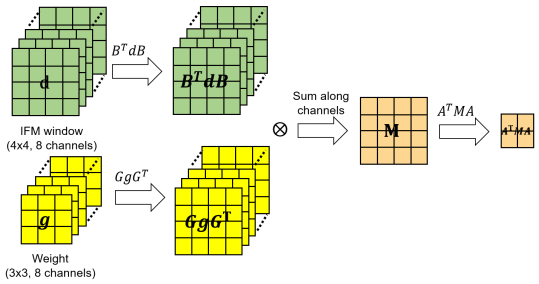


Fig. 1: Diagram for 2D multi-channel F(2x2, 3x3) Winograd Convolution.

For 3x3 CONV with stride 1, each output pixel is derived from a 3x3 window of the input feature map (IFM). WinoConv processes a 4x4 IFM window by handling the four 3x3 sub-windows in parallel. It exploits shared weights and IFM spatial overlap to restructure computation, reducing the number of required multiplications from 36 to 16 [15]. While WinoConv reduces computational complexity in CONV layers, it suffers from significant drawbacks. First, the 2x2 output of WinoConv requires splitting into respective rows, incurring buffer space and latency overhead. Next, the stride of 2 in WinoConv leads to inconsistencies between the throughput rate of IFM read from DRAM and line buffer output. To address these issues, we propose a novel data reordering format that is directly processed by our streaming NPU architecture.

III. Proposed Architecture

A. Hardware-Friendly Quantization

1) Binary Scaling: Adjusting values to powers of 2 is an ideal approach for hardware implementation. This is because multiplication can be replaced into shift operation which takes much less computational cost. Instead of setting all scaling factors to powers of two as in [18], our method simplifies the quantization process by ensuring that the ratio of scaling factors between consecutive layers is a power of two. Specifically, we set the ratio

$$\text{Scale}_{\text{Act}}^{(n+1)} / (\text{Scale}_{\text{Act}}^{(n)} \times \text{Scale}_{\text{weight}}^{(n)})$$

to be the power of two. This approach streamlines the quantization implementation by enabling efficient shift operations within the processing element (PE).

2) Adaptive Quantization: In our approach, we apply different quantization strategies based on the type of layer. For non-detection layers, we utilize Binary Scaling employing symmetric quantization in these layers to avoid complications that could arise from bias addition. On the other hand, detection layers—the final layers in the network—do not feed into subsequent layers. Therefore we employ asymmetric quantization in detection layers for higher granularity.

3) Layer Fusion: To mitigate the loss in quantization accuracy, one can reduce the number of quantization steps within a layer. Layer Fusion fuses BN operations into the weights and biases of a CONV layer, which are calculated as follows:

$$w_{fused} = w_{conv} \cdot \frac{\gamma}{\sqrt{Var[x] + \epsilon}}$$

$$b_{fused} = \frac{b_{conv} - E[x]}{\sqrt{Var[x] + \epsilon}} \cdot \gamma + \beta$$

w_{conv} and b_{conv} indicates the weight and bias of the CONV layer whereas γ and β indicates the weight and bias of BN layer. E[x] and Var[x] are defined as running mean and variance calculated over the batch during training. In hardware, the BN fused layer requires only one multiplication and one addition, thereby minimizing the latency in the data path.

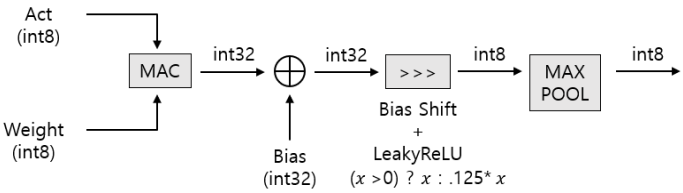


Fig. 2: Proposed Quantization Scheme

B. Overall Model Architecture

Fig. 3 presents the overall block diagram of the proposed design. The quantized weights and bias of the model are stored entirely in DRAM, whereas the input to each layer is given as stated in our streaming method.

C. Optimization of Data Access Patterns for Enhanced Reusability

To address the previously mentioned drawbacks of WinoConv-based hardware, improving data throughput and its consistency is crucial. Thus, we first introduce a feature map reordering technique, termed row stacking, designed to eliminate the overhead involved in merging rows to achieve a stride of 2. Also, we propose an optimized data access pattern,

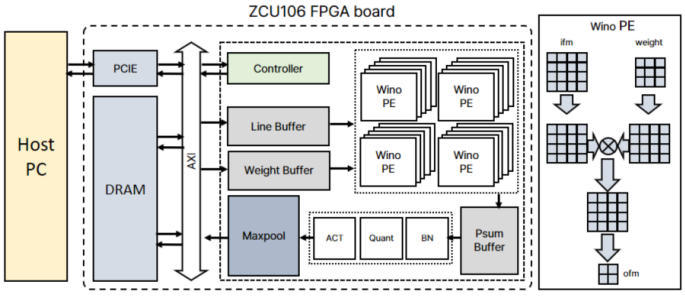
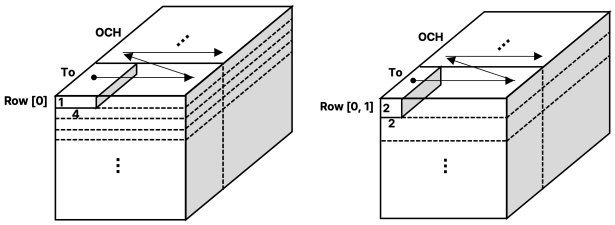


Fig. 3: Overview of the proposed architecture



(a) (b)

Fig. 4: Output Feature Map Pattern with 256-bit data blocks (a) Conventional Convolution (b) Winograd Convolution

known as horizontal pass [24], modified to obtain data from the row-stacked structure with minimal memory overhead.

Fig. 4 illustrates two types of OFM patterns: Fig. 4a depicts the OFM pattern of conventional CONV, whereas Figure 4b shows our proposed pattern for WinoConv. T_i and T_o refer to the tiled input and output channels, which are processed by the NPU concurrently as a group. In our row stacking method, after WinoConv, the output data block is stored without splitting its rows apart. In the subsequent layer, our line buffer can directly reformat this into a 4x4 IFM window.

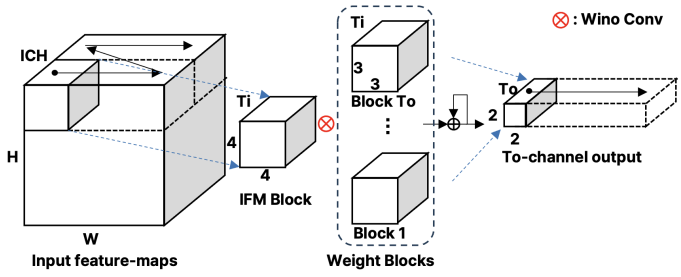


Fig. 5: Proposed input feature map block and weight block scheduling method for Winograd convolution

Fig. 5 illustrates the proposed horizontal pass method, which operates as follows: the line buffer output cube slides horizontally along the image width [24]. During each pass, the 4x4xT_i line buffer output cube shifts horizontally by a stride of 2 for every 2x2xT_i data block read from DRAM. Thus, the line buffer output synchronizes with DRAM read rate, thereby optimizing the pipeline and reducing memory requirements.

In each horizontal pass, the sliding IFM cube is convolved with T_o stationary filters, producing partial sums of size 2x2xT_o. We set T_i and T_o to the same value, so that accumulated partial sums can be directly sent to DRAM. Accounting for the extra rows needed for CONV padding and mitigating DRAM latency, the line buffer's memory utilization is:

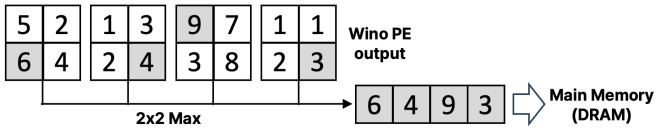
$$(2 \times 2 \times T_i) \times \left\lceil \frac{W_{ifm}}{2} \right\rceil \times \left\lceil \frac{\text{ich}}{T_i} \right\rceil \times (4 \text{ buffers})$$

D. Concat and Upsample layers

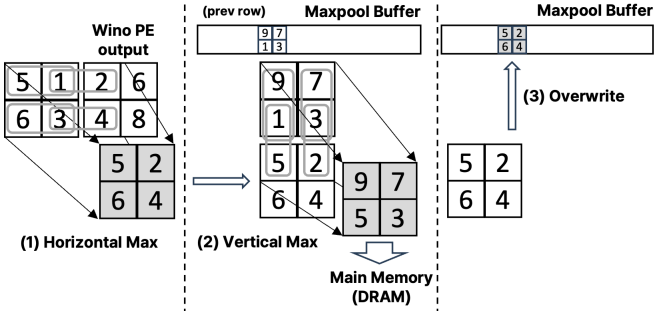
The concat layer combines output feature maps from previous layers along the channel axis for input to the next layer. Our NPU processes each feature map sequentially with horizontal passes, alternating read requests between feature maps by row. Upsample layers expand each pixel into a 2x2 grid, doubling the width and height of the OFM. This matches our row stacking scheme, keeping the output shape at 2x2xTo.

E. Max Pooling for strides 1 and 2

Since WinoConv outputs a 2x2 window to the MaxPool layer, pooling can be applied immediately without needing to wait for the next row of pixels. Therefore, on-chip buffer access is reduced, lowering power consumption. Fig. 6 illustrates the MaxPool layers of strides 1 and 2 in YOLOv3Tiny.



(a) MaxPool 2x2 Stride 2 Dataflow



(b) MaxPool 2x2 Stride 1 Dataflow

Fig. 6: MaxPool dataflow for 2x2xT_o activation tiles

6a depicts the MaxPool operation with a stride of 2, where four pooled values are concatenated to form a 1x4xTo data block, which is subsequently sent to the main memory. Fig. 6b demonstrates how a MaxPool with a stride of 1 can generate an output data shape of 2x2xTo in three steps:

1) *Horizontal Max*: Two adjacent 2x2 activation windows are analyzed, and the maximum value is computed horizontally across corresponding pixels.

TABLE I:
PERFORMANCE COMPARISON TO OTHER YOLOv3-TINY FPGA ACCELERATORS

	[10]	[11]	[12]	[13]	This work		[18]	This work
Model	YOLOv3-tiny							
Dataset	COCOval5k	Custom VOC2007	N/A	N/A	VOC2007+2012		Custom	
Year	2020	2021	2021	2020	2024		2023	2024
Input Width	416	448	416	416	416		320	320
Quantization	16b	8b	16b	18b	8b		8b	8b
Accuracy drop	2.5%p mAP	2.0%p mAP	N/A	N/A	2.78%p mAP		2.01%p mAP	0.93%p mAP
FPGA	ZYNQ-7020	Ultra96 V2	ZYNQ-7035	Virtex-7 VC707	ZYNQ ZCU106		Nexys A7-100T	ZYNQ ZCU106
Board Cost	$589	$249	$1499	$5244	$3234		$265	$3234
Clock	100MHz	250MHz	100MHz	200MHz	125MHz	200MHz	100MHz	125MHz
BRAM	185	248	248	141	105	105	185	105
DSP	160	242	485	2304	1053	1053	240	1053
LUT	25.9k	27.3k	N/A	48.6k	144.3k	136.0k	50.2k	144.3k
FF	46.7k	38.5k	N/A	93.2k	200.1k	198.0k	58.1k	200.1k
Latency	532ms	121ms	192ms	12.08ms	28ms	18ms	13.03ms	5.13ms
GOPS	10.45	31.50	28.99	460.8	232.72	362.01	95.08	292.18
Power	3.36W	4.26W	3.71W	4.81W	1.570W	2.317W	2.203W	1.570W
GOPS/DSP	0.0653	0.130	0.0598	0.200	0.227	0.354	0.396	0.285
GOPS/W	3.11	7.40	7.81	95.80	148.23	156.24	43.16	186.1
MAC Utilization	32.65%	50.00%	29.89%	50.00%	45.87%	44.60%	82.53%	58.65%

2) *Vertical Max*: The result buffered from the previous row is used to perform a vertical maximum operation, after which the 2x2 output is directly written to DRAM.
3) *Update Buffer*: The horizontal maximum values from the current row are used to update the values in the MaxPool row buffer, replacing the previous row's data.

IV. Experiment and Results

A. FPGA Implementation

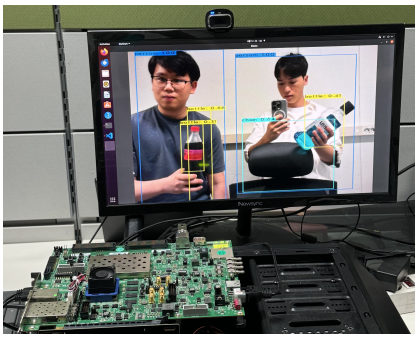


Fig. 7: Live demo of the proposed work on the ZCU106 FPGA board.

1) System Design and Implementation: The architecture was implemented on Verilog HDL and deployed on a Xilinx Zynq+ ZCU106 board operating at 125MHz and 200MHz. The FPGA board interfaces with a host PC via PCI Express, enabling transfer of weights and input images to the FPGA. The diagram of our proposed architecture is shown in Fig. 3.

2) Memory Utilization: Our architecture significantly reduces BRAM usage by utilizing only 105 blocks of 36K BRAM, notably smaller compared to [17]. This is achieved by a pipelined design and horizontal pass scheduling in our dataflow. As a result, the required on-chip memory is proportional to the product of width and channels. Consequently, our architecture is well-suited for deployment on smaller boards due to using less memory and, thanks to WinoConv, less DSPs.

3) Latency: WinoConv significantly reduces latency, and our data reordering scheme further decreases execution time by eliminating overhead. Our work on the custom YoloV3Tiny model takes 5.134 miliseconds to infer a single image, enabling real-time inference for 60 FPS or higher.

4) DSP Utilization: Our design utilizes 1,053 multipliers, 60% of the DSPs on the ZCU106, leaving room for future DSP utilization improvement. In comparison with the results of [13], our design achieved approximately 1.5 times the latency while using less than half the DSPs and 75% of BRAM.

5) Energy: Our NPU IP alone consumes 2.317W at 200MHz and 1.57W at 125MHz, comparable to small-scale boards such as [18]. The impressive efficiency in terms of GOPS/W is largely due to the use of WinoConv. It boosts GOPS while keeping the power consumption in check, leading to power-efficient processing.

B. Quantization Experiment and Results

For INT8 quantization, selecting an optimal bias shift value n is crucial for accurately capturing activation granularity in the detection layer. Among clipping thresholds set at approximately 1σ, 2σ, 3σ, and 4σ, we found clipping at the 3σ range showed the least precision degradation, achieving 83.18%.

Experimental results over the custom dataset reveal that over 98% of the data in the final detection layer falls within the 3σ range. Given that $2^{13} < 3\sigma(= 16386) < 2^{14}$, the optimal bias shift was determined to be 7 (15-8, considering the signed bit). After successfully applying this approach to other detection layers, we achieved a mean average precision (mAP) of 83.18% for the custom dataset, with precision reduced by only 0.93% compared to the original floating-point model.

V. Conclusion

We present a CNN accelerator based on Winograd Convolution, trained on a custom dataset for multi-camera vision

purposes. Our row stacking method significantly reduces memory and latency overhead compared to conventional dataflow, resulting in 5.13ms inference and 292.18 GOPS. Also, we achieved a 0.93% decrease in accuracy through our INT8 quantization strategy. Since our method fits well into streaming architectures, it will have a positive impact on the throughput of real-time object detection systems such as unmanned stores.

ACKNOWLEDGMENTS

This work was supported in part by the Technology Innovation Program (or Industrial Strategic Technology Development Program – No. 20014490, Development of Technology for Commercializing Lv.4 Self-driving Computing Platform Based on Centralized Architecture) funded by the Ministry of Trade, Industry & Energy (MOTIE, Korea) and in part under the artificial intelligence semiconductor support program to nurture the best talents (IITP-2023-RS-2023-00256081) grant funded by the Korea government (MSIT). This research was also supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2024-2020-0-01461) supervised by the IITP (Institute for Information & communications Technology Planning & Evaluation).

REFERENCES

[1] Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation." 2015. arXiv preprint arXiv:1505.04597.

[2] Y. Cheng, D. Wang, P. Zhou, et al. "Model Compression and Acceleration for Deep Neural Networks: The Principles, Progress, and Challenges," IEEE Signal Processing Magazine, vol. 35, no. 1, pp. 126–136, Jan. 2018(DOI: 10.1109/msp.2017.2765695)

[3] S.-J. Horng and P.-S. Huang, "Building unmanned store identification systems using YOLOv4 and Siamese network," Appl. Sci., vol. 12, no. 8, p. 3826, Apr. 2022.

[4] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang, and H. Yang, "Going Deeper with Embedded FPGA Platform for Convolutional Neural Network," FPGA '16, pp. 26–35, ACM, 3 2016

[5] Guo, Kaiyuan, Shulin Zeng, Jincheng Yu, Yu Wang, and Huazhong Yang. "[DL] A Survey of FPGA-based Neural Network Inference Accelerators." ACM Transactions on Reconfigurable Technology and Systems, vol. 12, no. 1, March 2019, article 2, 26 pages. Association for Computing Machinery, New York, NY, USA. DOI: 10.1145/3289185.

[6] Y. Ma, N. Suda, Y. Cao, J. Seo and S. Vrudhula, "Scalable and modularized RTL compilation of Convolutional Neural Networks onto FPGA," 2016 26th International Conference on Field Programmable Logic and Applications (FPL), Lausanne, 2016, pp. 1-8.

[7] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep Convolutional neural networks," in Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, pp. 161– 170, 2015

[8] Zeiler, Matthew D., and Rob Fergus. "Visualizing and Understanding Convolutional Networks." 2013. arXiv preprint arXiv:1311.2901.

[9] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.

[10] Z. Yu and C.-S. Bouganis, "A parameterisable FPGA-tailored architecture for YOLOv3-tiny," in Proc. Int. Symp. Appl. Reconfigurable Comput., Toledo, Spain, Apr. 2020, 2020, pp. 330–344.

[11] T. Adiono, A. Putra, N. Sutisna, I. Syafalni, and R. Mulyawan, "Low latency YOLOv3-tiny accelerator for low-cost FPGA using general matrix multiplication principle," IEEE Access, vol. 9, pp. 141890–141913, 2021.

[12] Q. Xiong, C. Liao, Z. Yang, and W. Gao, "A method for accelerating YOLO by hybrid computing based on ARM and FPGA," in Proc. 4th Int. Conf. Algorithms, Comput. Artif. Intell., Dec. 2021, pp. 1–7.

[13] A. Ahmad, M. A. Pasha, and G. J. Raza, "Accelerating tiny YOLOv3 using FPGA-based Hardware/Software co-design," in Proc. IEEE Int. Symp. Circuits Syst. (ISCAS), Oct. 2020, pp. 1–5.

[14] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer CNN accelerators," in Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO), Oct. 2016, pp. 1–12

[15] A. Lavin and S. Gray, "Fast Algorithms for Convolutional Neural Networks," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 4013-4021, doi: 10.1109/CVPR.2016.435.

[16] Cheng, Yu, Duo Wang, Pan Zhou, and Tao Zhang. "A Survey of Model Compression and Acceleration for Deep Neural Networks." 2020. arXiv preprint arXiv:1710.09282.

[17] Z. Zhang, Z. Li and H. Chen, "A cache structure and corresponding data access method for Winograd algorithm," IET International Radar Conference (IET IRC 2020), Online Conference, 2020, pp. 634-639, doi: 10.1049/icp.2021.0723.

[18] Minsik Kim, Kyoungseok Oh, Youngmock Cho, Hojin Seo, Xuan Truong Nguyen, Hyuk-Jae Lee, "A Low-Latency FPGA Accelerator for YOLOv3-Tiny With Flexible Layerwise Mapping and Dataflow," IEEE, Dec. 2023, pp. 1158-1171

[19] P. Adarsh, P. Rathi and M. Kumar, "YOLO v3-Tiny: Object Detection and Recognition using one stage improved model," 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India, 2020, pp. 687-694, doi: 10.1109/ICACCS48705.2020.9074315.

[20] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. (Mar. 2016). "XNOR-Net: ImageNet classification using binary convolutional neural networks." [Online]. Available: https://arxiv.org/abs/1603.05279

[21] 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.

[22] Y. Kim, J.-S. Choi, and M. Kim, "A real-time convolutional neural network for super-resolution on FPGA with applications to 4K UHD 60 fps video services," IEEE Trans. Circuits Syst. Video Technol., vol. 29, no. 8, pp. 2521–2534, Aug. 2019.

[23] V. Munasinghe, T. -H. Lee, H. -J. Lee, T. Sung Kim and J. -S. Kim, "Multi-Camera-Based Product Recognition for Inventory Management," 2023 International Conference on Electronics, Information, and Communication (ICEIC), Singapore, 2023, pp. 1-3, doi: 10.1109/ICEIC57457.2023.10049880.

[24] D. T. Nguyen, H. Je, T. N. Nguyen, S. Ryu, K. Lee and H. -J. Lee, "ShortcutFusion: From Tensorflow to FPGA-Based Accelerator With a Reuse-Aware Memory Allocation for Shortcut Data," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 69, no. 6, pp. 2477-2489, June 2022, doi: 10.1109/TCSI.2022.3153288.

[25] Xilinx. 8-Bit Dot Product Acceleration White Paper. Accessed June 27, 2017. Available at: https://docs.amd.com/v/u/en-US/wp487-int8-acceleration.

[26] J. Yu et al., "Instruction driven cross-layer CNN accelerator with winograd transformation on FPGA," 2017 International Conference on Field Programmable Technology (ICFPT), Melbourne, VIC, Australia, 2017, pp. 227-230, doi: 10.1109/FPT.2017.8280147.