A Real-Time Sparsity-Aware 3D-CNN Processor for Mobile Hand Gesture Recognition

Seungbin Kim
Graduate School of Artificial Inteligence
UNIST, Ulsan, Repulic of Korea
seungbin.leon.kim@unist.ac.kr

Jueun Jung
Department of Electrical Engineering
UNIST, Ulsan, Repulic of Korea
jueunjung@unist.ac.kr

Wuyoung Jang
Graduate School of Artificial Inteligence
UNIST, Ulsan, Repulic of Korea
wuyoungjang@unist.ac.kr

Hoichang Jeong
Department of Electrical Engineering
UNIST, Ulsan, Repulic of Korea
jhc3261@unist.ac.kr

Kyuho Lee
Dept. of EE / Graduate School of AI
UNIST, Ulsan, Repulic of Korea
kyuho.jsn.lee@unist.ac.kr

***Abstract*—A sparsity-aware 3D convolution neural network (CNN) accelerator is proposed for the real-time mobile hand gesture recognition (HGR) system. The complex computation of 3D convolution with the video data makes it difficult for real-time operation, especially in the resource-constrained mobile platform. To facilitate real-time implementation of HGR, this paper proposes two key features: 1) the inter-frame differential aware input method and IDANet to reduce the MAC operation and the external bandwidth by 84% and 95.2%, respectively, and achieve 79.97% accuracy on NvGesture dataset; 2) a low-latency 3D-CNN accelerator that utilizes activation and weight sparsity, achieving 31× faster inference latency than the state-of-the-art. The proposed processor is designed in 65 nm CMOS technology. It consumes 262 mW of power and achieves 599 GOPS/W of energy efficiency. As a result, the system realized 1.584 ms inference latency for real-time HGR in a mobile platform.**

Keywords—3D-CNN, hand gesture recognition processor, VLSI

I. Introduction

As intelligent devices have become essential in the daily life of human beings recently, a more intuitive and convenient interface is required such as voice recognition [1], action recognition [2]-[3], and hand gesture recognition (HGR) [4]-[7]. Among them, HGR is robust to surrounding noise and it takes less physical space, thus, HGR is widely adopted as a control interface in head-mounted displays [4] and in-vehicle infotainment [5].

The combination of traditional 2D-CNN and LSTM [8] is frequently used for action recognition and HGR for their high accuracy by utilizing spatio-temporal coherence of consecutive image frames. These approaches individually processed spatial data and temporal data. However, they suffered from relatively poor accuracy in recent mega-scale datasets such as UFC101 and Sports-1M. 3D-CNN that uses a stack of multiple subsequent frames according to the input duration outperforms with such datasets. As shown in Fig. 1, 3D-CNN moves kernels not only in spatial direction but also in temporal direction to simultaneously consider spatio-temporal coherences. Several works attempted to utilize 3D-CNN in HGR [6]-[7]. However, 3D-CNN involves more massive MAC operations and memory footprint compared to the conventional 2D-CNN. For example, HGR with 3D-ResNet under 112×112 resolution and 16-frames input duration requires 11.4 G MAC operations and 81.5 MB memory (leads to 51.5 GB/s), which are 13.04× and 3.44× compared to the conventional ResNet-18, respectively. Moreover, it requires up to 13.5 MB of weight and 1.53 MB of input/output memory for just one layer that cannot be stored on-chip SRAM at once in mobile devices. Therefore, a realization of real-time 3D-CNN was not feasible on mobile devices; e.g. implementation on Qualcomm's Snapdragon 865 showed 2.5 s of inference latency [9].

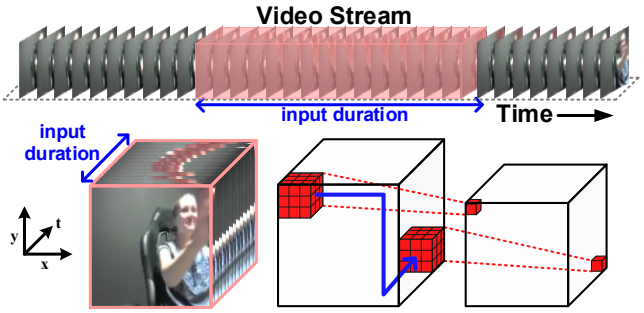


Fig. 1. 3D-CNN architecture with inter-frame spatio-temporal coherency.

Previous CNN accelerators [10], [11] were not possible to compute 3D-CNN or required redundant external memory accesses for activation because they did not consider datapath for 3D convolutions. A hierarchical cache system for 3D-CNN [12] was designed to resolve the issue. Although it achieved high energy efficiency with FPGA implementation, very large on-chip memory (1.79 MB) was required. Another 3D-CNN accelerator with Winograd convolution was proposed for fast inference in [13]. However, it was not compatible with other 3D-CNNs because its Winograd computation method was totally different from normal 3D-CNNs.

To solve the abovementioned challenges of 3D-CNN, this paper proposes a Hardware-Algorithm co-optimization to maximally utilize activation and weight sparsity. The algorithm proposes the inter-frame differential aware (IDA) input method and a neural network architecture optimized for IDA, namely IDANet. For hardware optimization, a dedicated architecture is proposed that is capable of zero-tile skipping as well as Region-of-Interest(ROI)-only computation, which can fully utilize *activation sparsity* manifested during algorithm optimization. In addition, the proposed PE architecture and sparsity-distribution-aware workload allocation utilize *weight sparsity* to increase processing speed.

The rest of this paper is organized as follows. Section II introduces the proposed algorithm optimization process. Section III describes the proposed hardware and its operation. Implementation results and conclusion will be followed in Section IV and V, respectively.

II. The Proposed IDANet Optimization

Previous works for 3D-CNN attempted to divide the 3D kernels into 2D spatial kernels and 1D temporal kernels [2], or

into 2D kernels for three directions [3]. But they cannot be used in mobile devices without reducing the massive computation burden of 3D-CNN. To reduce computation complexity and improve energy efficiency, IDA input method is suggested.

Key idea is to increase activation sparsity that can be skipped during computation by utilizing spatio-temporal coherence of video stream. The spatial coordinate of hand gestures in 2D images changes in the time domain. Hence, unchanged pixel values between frames do not contain motion information while only the pixels whose values are changed contain meaningful information about hand gestures. Based on the characteristic of HGR data, IDA input focuses on the inter-frame changes by using differential images of two subsequent images.

Fig. 2 shows the data formation of IDA input method with single-channel depth images for noise-tolerant HGR. The reference frame takes the original image and the following 15 frames of input duration are differential images that possess enormous sparsity. It facilitates faster and more energy-efficient inference if accelerated with sparsity-aware hardware that can skip zero values. A rectangular ROI is set to hold inter-frame change values, i.e. hand motion. The background pixels are zero and they are not fetched into the accelerator for efficiency. Only ROI pixels are fetched from the external memory and executed for 3D convolution. IDA input method increases *input data sparsity* to 94.7%.

The proposed IDANet is described in Fig. 3. NVGesture dataset for touch-less driver control [7] is used for training, and 3D-ResNet-18 [6] is used as a skeleton model that showed outstanding performance compared to 2D-CNN with the mega-scale dataset. Although the IDA input method increased activation sparsity, it is important to maintain the sparsity in each layer for fast processing. However, batch normalization (BN) of the skeleton disturbs the full propagation of activation sparsity. Experiment shows that activation sparsity reduces from 94.7% to 39.7% after passing the 1st BN. The sparsity becomes severely ruined with deeper BN layers. Thus, BN layers are removed in IDANet to avoid sparsity vanishing, and 82.3% of activation sparsity over the whole network is maintained.

Since enough features are extracted after Layer-2 with the highly-sparse IDA input data format, IDANet replaces Layer-3 and Layer-4 with MaxPool layers. Then, global pruning is successfully applied. Finally, IDANet removes 94% parameters of the skeleton model. The proposed IDANet shows 2.29% accuracy degradation compared to the skeleton with raw input, but applying ROI-Only computation reduces 95.2% of the external memory accesses and 84% of MAC operations, on average; facilitating real-time inference in mobile processors.

III. Hardware Architecture

A. Overall Architecture

Fig. 4 shows the overall architecture of the proposed processor. It consists of a global scheduler, an activation shared register (ASR), 8 PE Arrays (PEAs), a corner detector, and a global scheduler. Each PEA contains 49 weight sparsity-aware PEs (WSAPEs), a weight memory (WMEM), and a PE controller.

The ASR loads an input tile that is divided into 49 sets of 3×3×3 tensors, and each of them is transmitted into

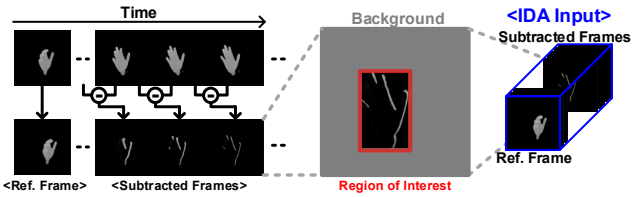


Fig. 2. Data formation of the proposed IDA input method.

Layer Name	IDANet	Output Size
Conv1	7x7x7, 64, stride(1,2,2)	16x56x56x64
MaxPool	2x2x2, 64, stride(2,2,2)	8x28x28x64
Layer-1	[3x3x3, 64, stride(1,1,1); 3x3x3, 64, stride(1,1,1)] x2	8x28x28x64
Layer-2	[3x3x3, 128, stride(2,2,2); 3x3x3, 128, stride(1,1,1)] x2	4x14x14x128
MaxPool	2x2x2, 256, stride(2,2,2)	2x7x7x256
MaxPool	2x2x2, 512, stride(2,2,2)	1x4x4x512
Average Pool, 512-26d FC		

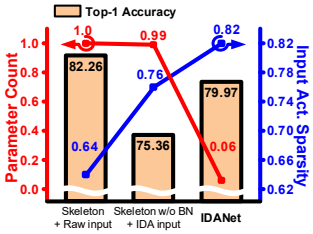


Fig. 3. The IDANet architecture and comparison.

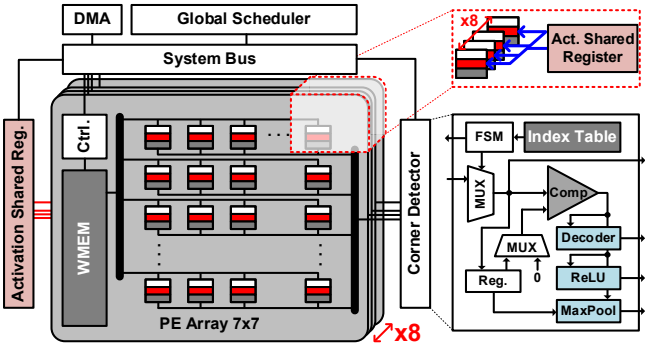


Fig. 4. Overall architecture of the proposed processor.

WSAPEs in a single PEA for tile-parallel processing in Fig. 5(a). The WSAPEs on the same location of each PEA shares identical activation to calculate 1×7×7 output feature map of 8 output channels (*OCs*) in parallel as depicted in Fig. 4. Each PEA completely computes the whole *OCs* for a given input tile.

8 PEAs share a corner detector whose roles are 1) storing output feature maps into the external memory; 2) searching the ROI corner information of output feature map; and 3) performing ReLU and MaxPool operations. Its FSM controls datapath from PEA to DMA by referring to the Index Table that contains the order of *OC* to be computed in each PEA. For corner detection, the pixel values of the output feature map are compared with "0" to extract the ROI corner information. The decoder searches the location of the corner pixel from the edge of each feature map where the comparator output becomes "1" at the corner pixels. The location of the corner pixels is transformed into a coordinate index and sent to the global scheduler. ReLU is performed by replacing "0" values according to the comparator output. The corner detector loads rows of output feature maps from PEA. Output feature map of the adjacent row is temporarily stored in the register and used as input to the comparator to support MaxPool operation.

B. Activation-Sparsity-Aware Global Scheduler

The ROI has different corner information with respect to time as hand gesture moves in both spatial and temporal domains. The global scheduler gathers corner information of each frame over the entire time domain and input channels (*ICs)* beforehand, then, decides the ROI tensor for activation. The ROI corner information is represented as (x, y)[IC_N, t_M], where IC_N and t_M are N^{th} *IC* and M^{th} frame in the time domain, as shown in Fig. 5(b).

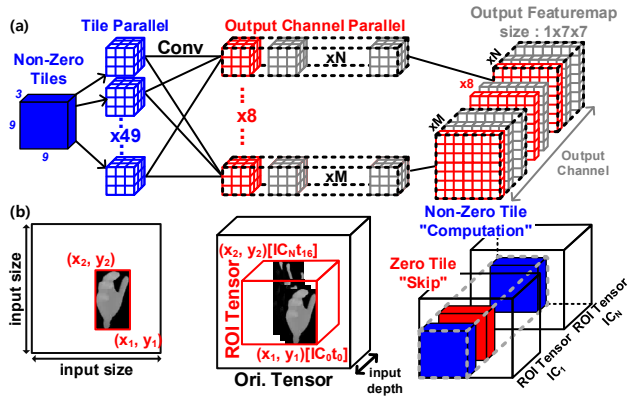


Fig. 5. (a) Computation parallelism and (b) operation of activation-sparsity-aware global scheduler.

The average size of the ROI tensor for each layer is 113 kB. It is too large to be fetched into on-chip memory at once in mobile devices. Therefore, ROI tensors are divided into input tiles for computation. The ROI tensor involves Zero-Tile whose values are only 0's because corners at each frame vary by the time domain and *IC*. The computations of Zero-Tiles are also skipped. Thus, the global scheduler hierarchically skips zero-value computations of both outside (background) and inside (Zero-Tile) the ROI tensor. Each tile is computed in order of [*time domain – IC – spatial domain*] direction. This enables ASR to reuse temporal data. Also, complete output activation is obtained by loading input activation once.

The global scheduler supports versatile sizes of CONV/FC kernel and stride. Some large CONV kernels are divided into 3×3×3 kernels for convolution and accumulated through inter-PE datapath. The input tile is enlarged to make 49 tensors when the stride is bigger than 1. For FC layer, the global scheduler transforms matrix multiplications into 27 vector multiplications for WSAPEs.

C. Weight-Sparsity-Aware PE and WMEM

The size of weight parameters of IDANet is 3.85 MB that is not possible to store on chip. However, the required memory size can be dramatically reduced if the sparse data caused by pruning is compressed. Previous compression methods for 2D-CNN [10] cannot be utilized for 3D tensors because they only expressed 2D coordinates of non-sparse values. Fig. 6(a) explains the proposed 3D-tensor-coordinate compression for weight-sparsity-aware computation.

Weight kernels' non-sparse value (W-value), kernel mask (W-mask), and coordinate index (W-index) are stacked in WMEM as 1D vector. W-index indicates the coordinate of elements in a kernel. The 3D coordinates of 3×3×3 tensor are with the same order of input pixels stored in ASR; resulting in no additional decoding process for vector multiplication. Also, data loaded from the WMEM must be identified which kernel it belongs to. W-mask and boundary indicator are used for the identification. W-mask has binary values that indicate the first non-zero value in the kernel to distinguish kernel boundaries. Boundary indicators are placed between the two nearby kernels that are multiplied with different input activation to make WSAPE request new input activation to ASR. The boundary indicator in W-mask is *1* if the two kernels are from the same layer. The proposed 3D-tensor-coordinate compression method reduces 94.9% of the required size of WMEM.

Fig. 6(b) explains the operation of WSAPE. It selects one data among 27 input activations of ASR where the coordinate

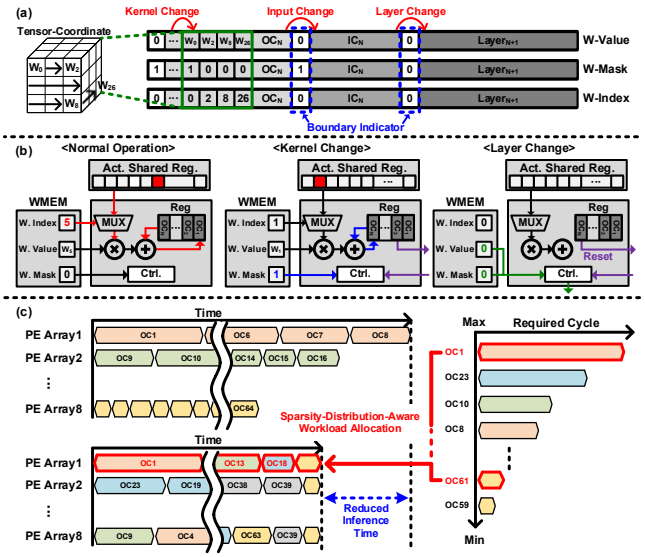


Fig. 6. (a) 3D-tensor-coordinate compression for WMEM, (b) architecture and operation of weight sparsity aware PE, and (c) sparsity-distribution-aware workload allocation.

of selected data is identical to W-index of corresponding weight and performs MAC operation every cycle. The registers in PE are used for partial sum accumulation. PE controller changes the register address when it detects kernel change (W-mask = 1). When the PE controller detects a boundary indicator, it stops MAC operation and requests ASR for a new input tile. The registers are refreshed for the next layer computation after the controller finishes sending the computed output pixel to the corner detector at the layer change.

The weight sparsity varies over the entire channels. Weight parameters are stored in 8 WMEMs obeying sparsity-distribution-aware order. Partial sum accumulation involves complex data routing if one output activation is computed in different PEAs or the sequence of output activation being computed within a PEA is different. Therefore, weights are aligned to have identical *OC* computing order over the entire *IC*s. Ignoring sparsity distribution degrades PE utilization and harms inference latency due to unbalanced workloads. To prevent such degradation, kernel order is adjusted before being stored in WMEM by sparsity-distribution-aware workload allocation in Fig. 6(c). Since the required cycles for each *OC* differ depending on the number of non-sparse elements in the corresponding weight kernel, computation queues for *OC*s are sorted and allocated to balance the workloads for minimum inference latency. The proposed sparsity-distribution-aware workload allocation maximizes the utilization of WSAPE and increases the processing speed by 19.8×.

IV. Implementation and Measurement

Fig. 7 shows measurement results with NvGesture dataset that contains 26 classes of hand/finger gestures. Training of IDANet using only differential images failed since it could not distinguish fingers clearly over the entire input duration. The reference frame in IDANet makes it trainable by providing spatial information to distinguish hand gestures for all classes. It achieves 79.97% accuracy on average, and the test results of several classes are shown in the right figure.

Fig. 8 shows the result of the external memory bandwidth reduction and sparsity of the proposed IDANet. The peak memory bandwidth of the skeleton model requires 34.9 GB/s,

which cannot be supported by DDR4. The proposed processor utilizes 82.3% of activation sparsity generated by IDA input method with ROI-Only computation, reducing the external bandwidth by 95.2%.

Fig. 9 shows the layout photograph and specification of the proposed processor. It is designed using Synopsys Design Compiler and IC Compiler with 65 nm CMOS Logic Process. It consumes 262 mW under 1.1 V @ 200 MHz and occupies 12 mm^2 with 198 kB on-chip SRAM. The processor executing IDANet achieves 156.8 GOPS and 599 GOPS/W of peak performance and energy efficiency, respectively. Table I shows comparisons with the state-of-the-art 3D-CNN processors. Short latency is the crucial metric in order for HGR system to be used as a seamless and convenient interface. By deploying both activation and weight sparsity-aware computations, the proposed processor is the fastest 3D-CNN accelerator with 1.584 ms of inference latency, which is a 31× improvement over [13].

V. Conclusion

A sparsity-aware 3D-CNN accelerator and Inter-Frame Difference Network (IDANet) are proposed for real-time mobile hand gesture recognition by hardware-algorithm co-optimization. With Region-Of-Interest-Only computation supported by the proposed activation sparsity-aware global scheduler, IDANet reduces MAC operation and the external bandwidth by 84% and 95.2%, respectively, with only 2.29% accuracy degradation with NvGesture dataset. Thanks to the weight sparsity-aware PE and sparsity-distribution-aware workload allocation, the proposed processor achieves a 19.8× faster processing speed. The entire system improves system latency of state-of-the-art 3D CNN accelerator by 31×. As a result, a real-time HGR system with 599 GOPS/W energy efficiency and 1.584 ms inference latency was successfully implemented.

Acknowledgment

This research was supported in part by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2020-0-01847) supervised by the IITP (Institute of Information & Communications Technology Planning & Evaluation), and in part by Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (2021-0-00871, Development of DRAM-Processing-In-Memory Chip for DNN Computing), and in part by Samsung Electronics. The EDA tool was supported by the IC Design Education Center (IDEC), Korea.

References

[1] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh and K. Shaalan, "Speech Recognition Using Deep Neural Networks: A Systematic Review," in *IEEE Access*, vol. 7, pp. 19143-19165, 2019.

[2] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun and M. Paluri, "A Closer Look at Spatiotemporal Convolutions for Action Recognition," *IEEE/CVF CVPR*, 2018, pp. 6450-6459.

[3] A. Stergiou and R. Poppe, "Spatio-Temporal FAST 3D Convolutions for Human Action Recognition", *IEEE ICMLA,* 183-190, 2019.

[4] S. Choi, J. Lee, K. Lee and H.-J. Yoo, "A 9.02mW CNN-stereo-based real-time 3D hand-gesture recognition processor for smart mobile devices," *IEEE ISSCC*, 2018, pp. 220-222.

[5] F. Parada-Loira, E. González-Agulla and J. L. Alba-Castro, "Hand gestures to control infotainment equipment in cars" *IEEE IV Symp.*, 2014, pp. 1-6.

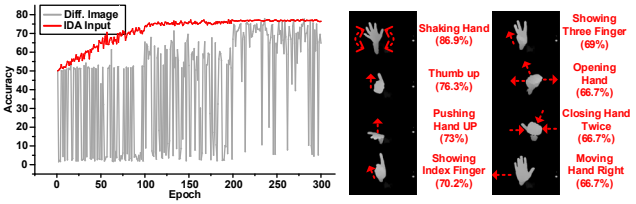


Fig. 7. Accuracy of IDANet and evaluation result with NvGesture dataset.

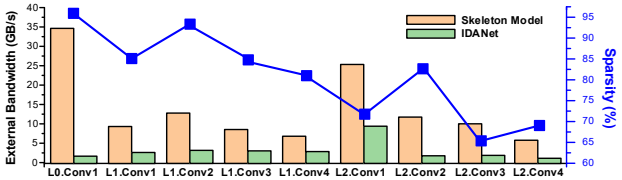
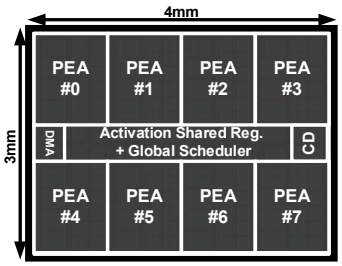


Fig. 8. External bandwidth reduction and sparsity through layers.



Process	65nm CMOS
Chip Size	4 mm x 3 mm
SRAM	198 kB
V_{DD}	1.1 V
Frequency	200 MHz
Power	262 mW
Peak Perf.	157 GOPS
Power Efficiency	599 GOPS/W

Fig. 9. Photograph of the chip layout and performance summary.

TABLE I. Comparison Table

	FPL'17 [14]	MICRO'18 [12]	ISFPGA'18[13]	DATE'20 [15]	*This Work*
Implementation	Xilinx ZC706	32nm CMOS	Xilinx VSU440	Xilinx VC707	65nm CMOS
Precision	16-bit fixed	8-bit fixed	16-bit fixed	8-bit fixed	16-bit fixed
Sparsity Awareness	N/A	N/A	N/A	N/A	Activation Weight
Performance [GOPS]	142	192	784.7	714	156.8
Power Eff. [GOPS/W]	7.94	N/A	30.18	N/A	599
On-chip SRAM [MB]	N/A	1.79	N/A	4.5	0.198
Latency [ms]	5425	N/A	49.1	107.9	1.584

[6] O. Köpüklü, A. Gunduz, N. Kose and G. Rigoll, "Real-time Hand Gesture Detection and Classification Using Convolutional Neural Networks," *IEEE Int. Conf. on Automatic Face & Gesture Recognition*, 2019, pp. 1-8.

[7] P. Molchanov *et al*, "Online Detection and Classification of Dynamic Hand Gestures with Recurrent 3D Convolutional Neural Networks," *IEEE CVPR*, 2016, pp. 4207-4215.

[8] J. Ng *et al*, "Beyond short snippets: Deep networks for video classification" *IEEE CVPR*, 2015, pp. 2-7.

[9] Niu, Wei, et al. "RT3D: Achieving Real-Time Execution of 3D Convolutional Neural Networks on Mobile Devices." in *Proc. of the AAAI Conference on Artificial Intelligence (AAAI)*. Vol. 35. No. 10. 2021.

[10] L. Lu and Y. Liang, "SpWA: An Efficient Sparse Winograd Convolutional Neural Networks Accelerator on FPGAs," *ACM/ IEEE Design Automation Conference (DAC)*, 2018, pp. 1-6.

[11] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *ACM SIGARCH Computer Architecture*, vol. 44, pp. 367–379, 2016.

[12] K. Hegde, R. Agrawal, Y. Yao and C. W. Fletcher, "Morph: Flexible Acceleration for 3D CNN-Based Video Understanding," *IEEE/ACM MICRO*, 2018, pp. 933-946.

[13] J. Shen *et al*, "Towards a Uniform Template-based Architecture for Accelerating 2D and 3D CNNs on FPGA," *ACM/SIGDA FPGA*, 2018, pp. 97–106.

[14] H. Fan, X. Niu, Q. Liu and W. Luk, "F-C3D: FPGA-based 3-dimensional convolutional neural network," in *FPL*, 2017, pp. 1-4.

[15] T. Tian, X. Jin, L. Zhao, X. Wang, J. Wang and W. Wu, "Exploration of Memory Access Optimization for FPGA-based 3D CNN Accelerator," *IEEE/ACM DATE*, 2020, pp. 1650-165