Journal of Trends and Challenges in Artificial Intelligence

Vol. 02, No. 02 (2025) 37-42, DOI: 10.61552/JAI.2025.02.001 - http://jai.aspur.rs

DESIGN AND DEVELOPMENT OF AN FPGA-BASED HARDWARE ACCELERATOR FOR IMPROVING COMPUTATIONAL PERFORMANCE

N.Gomathi[1]
M. Jayasanthi

Received 16.06.2024.
Revised 21.07.2024.
Accepted 29.08.2024.

Keywords:

Hardware Accelerator, Convolutional Neural Network, Field Programmable Gate Arrays, Optimisation.

Original research

A B S T R A C T

Due to recent breakthroughs in digital technology and the accessibility of reliable data, a field of artificial intelligence called machine learning has arisen. This field has shown its capability and efficiency in addressing intricate learning issues that were previously unsolvable. Convolutional Neural Networks (CNNs) have shown to be very successful in applications involving picture identification and recognition. The demanding nature of these tasks requires substantial Central Processing Unit (CPU) processing power and storage bandwidth, which conventional CPUs cannot provide to meet the needed performance benchmarks. Therefore, hardware accelerators using Field Programmable Gate Arrays (FPGAs) have been used to enhance the efficiency of CNNs. FPGAs have been lately used to strengthen the execution of machine learning networks because of their capacity to optimize parallelism and their energy economy. This paper proposes a new composite hardware design for a CNN accelerator to tackle these difficulties. It is called a CNN-based FPGA Hardware Accelerator (CNN-FPGA-HA). The FPGA-based accelerator has clear benefits due to its programmable nature, adaptability, power efficiency, and ability to do enormous parallel processing with CNN. Based on the experimental findings, the fully pipelined CNN hardware accelerator obtains an effectiveness of 328.32 GOP/s, with a power consumption of 42.31 GOP/s/w. This indicates that CNN-FPGA-HA surpasses the outcomes of prior techniques.

1. INTRODUCTION

Convolutional Neural Networks (CNNs) have shown exceptional ability in abstracting features, a crucial stage in image processing (Pinaya et al., 2020). They are extensively used in diverse fields, such as medical image identification, autonomous driving, and more. The demanding Multiply-Accumulate (MAC) processes needed by CNNs make it difficult for general-purpose Central Processing Units (CPUs) to reach satisfactory processing speed, highlighting the need for specialized accelerators (Ju & Gu, 2022; Kang & Kim, 2022).

Graphical Processing Units (GPUs) are the dominant choice for accelerator architecture since they are user-friendly and provide exceptional performance. Their significant energy demands make them inappropriate for low-power situations. Application-Specific Integrated Circuit (ASIC) offers efficient data processing with low energy consumption, but their substantial expenses restrict their practicality for broad use by consumers (Petrou et al., 2022). The unchangeable circuit

[1] Corresponding author: Iuliia Vasilikhina
Email: elenapopkova@yahoo.com

configuration of ASICs renders them vulnerable to becoming outdated as neural network topologies progress. Field Programmable Gate Arrays (FPGAs) provide an attractive option by achieving a harmonious combination of power use, processing speed, and adaptable programming (Samantaray et al., 2021). FPGAs are well-suited for implementing neural networks due to their ability to adjust to changing network structures and comply with power limitations in different applications.

As a result, there is a notable disparity between the real-world performance obtained and the maximum potential set by the external storage bandwidth and processing capacity of FPGAs. This study presents a CNN-based FPGA Hardware Accelerator (CNN-FPGA-HA), a reconfigurable FPGA-based accelerator for CNNs that operates at higher accuracy and minimal power. CNN-FPGA-HA is specifically developed to overcome current limitations in CNN acceleration. The primary goal is to create a high-performance accelerator using FPGA technology specially designed for the Virtex-7 FPGA design. This endeavor seeks to use FPGAs' inherent parallel processing capacities to expedite intricate calculations, explicitly targeting applications such as image analyzing, machine learning, and signal analysis.

2. BACKGROUND AND SUMMARY

This section deals with CNN models used for compression and different FPGA-based accelerators.

2.1 CNN Methods

The current neural network architecture tends to get more complex by increasing its depth to improve accuracy, resulting in a significant duplication of parameters. To implement CNN forward inferences on edge devices with limited computation, storage, and electric power capabilities, two commonly used ways for compressing CNN models are used. These methods include system pruning, variable quantization, and explicitly training a smaller network.

Network pruning involves incorporating a dense network relationship into a sparse networking interaction, reducing redundant system variables and the complexity of the network (Li & Louri, 2021). Guo et al. (2021) devised a conventional technique for network pruning, whereby network connections with weights below a particular threshold are eliminated after the first training (Guo et al., 2021). The pruned system is then updated to improve accuracy. The outcomes of the trials show that this pruning approach yields a compression of 8 times and 12 times on AlexNet and VggNet, correspondingly. The drawback of this approach is that it necessitates storing and utilizing sparse tensors to accommodate sparse neural networks, and its unique calculation mechanism impedes parallel processing.

Another approach involves quantizing a network's weight variables and activating values to reduce bit widths to preserve storage space and computational capacity (Liang et al. 2021). More and more researchers are trying to replicate absolute floating-point values using lower fixed-bit widths. Some authors. used 8-bit fixed-point variable quantization to provide substantial speedup while maintaining high accuracy (Wang et al. 2020).

Creating a lightweight network topology directly circumvents the intricate process of pruning, measuring, retraining, and fine-tuning the system. Weight variables in Squeeze Net are reduced by compressing the amount of feature mapping channels using a Fire component. The model demonstrates categorization accuracy comparable to AlexNet, with a model size of just 1/50th that AlexNet (Zhu et al., 2020). MobileNet reduces the number of variables and calculations by about nine times using deep separable inversion (Bouguezzi et al., 2021). DiracdeltaNet was introduced as a more efficient alternative to traditional networks (Yang et al., 2022). This was achieved by substituting the 3 x 3 convolution with a shifting method to realign pixels, removing the need for multiplication operations while accepting a slight decrease in accuracy. According to the given research, it is strongly recommended to immediately create a small-scale network with a certain degree of accuracy.

2.2 FPGA-based accelerators

FPGAs often speed up computationally complex models because they provide flexible and highly parallel acceleration at a cheap development cost. Utilizing software-level programming techniques, such as High-Level Synthesis (HLS), simplifies the process for developers to implement CNN on FPGA (Curzel, 2024). This approach partially uses the FPGA's potential to be customized (Zhang et al., 2022). Resource utilization must be optimized to maximize the limited logic resources available to deploy CNN on FPGA. VGGNet has 136 million weights, occupying 512 MB of memory for feature storage (Wang et al. 2020). More than 35 Giga Operations Per Second (GOPS) are required to process each input picture. A sophisticated CNN accelerator must be meticulously constructed to meet the limited computational and storage demands.

The primary objective of implementing FPGA-based CNN is to enhance the computational efficiency of the convolutional layer by improving its operations. The research introduced an acceleration model that relies on a roofline-based CNN (Park et al., 2020). Considering the computational power and connectivity limitations, the revised model can accurately represent the maximum theoretical efficiency attained. A modular Register Transfer Language (RTL) compiler was suggested to incorporate HLS using Python (Huggi & Jamuna, 2020). It accelerates CNNs by optimizing loop unrolling for convolutional layer operations and effectively generates modularized Verilog RTL scripts for FPGA platforms. The accelerator utilizes the traditional Line buffer to create a specialized convolution hardware component to reduce resource use (Liu et al., 2021). An evolving precision quantized model has been suggested to get an appropriate level of quantization fidelity for the weighting of every layer in the system.

To effectively address the actual use of computer vision and the limitations of FPGA accelerator layout, particularly in the context of edge devices, an optimal FPGA accelerator must satisfy the following criteria: 1) The system must not have an excessive number of elements that are represented as low-precision fixed-point numbers, while still maintaining reliable precision. 2) The network's architecture should be compatible with hardware, allowing for efficient data scheduling and permission. 3) The network's computing functional mode should be easily deployable on FPGA.

3. PROPOSED FPGA-BASED HARDWARE ACCELERATOR

The Matrix Multiplication Generation (MMG) array performs numerous tasks in CNN, such as convolution, standardization, Rectified Linear Unit (ReLU), and pooling. The settings and input photos are kept in external memory. A ping-pong weight buffering is inserted into the MMG array and memory to optimize the bandwidth. Biases are stored in the entries of the MMG arrays. The feature mapping buffer holds interim mappings of features to prevent the delay caused by reading and writing data from off-chip storage. The accelerator is regulated using a universal Finite State Machine (FSM).

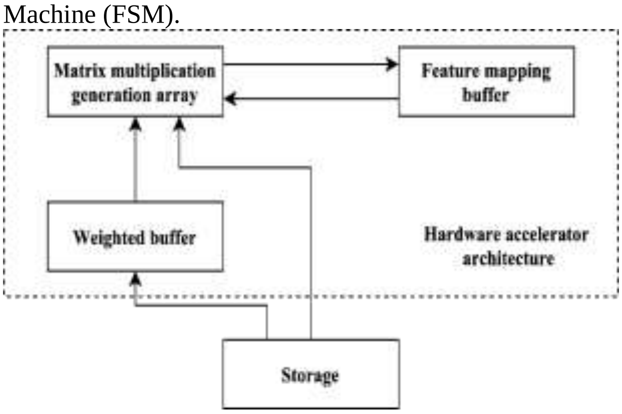

Figure 1. Hardware accelerator system

Figure 1 provides a comprehensive understanding of this accelerator via its block design.

A. System Architecture

The system comprises two primary components: a desktop CPU as the host and an accelerator FPGA board as the development substrate. The FPGA acceleration board seamlessly fits into the desktop CPU's Peripheral Component Interconnect standard (PCI) slot. The logic on the FPGA chip is categorized into two distinct components based on their respective functions: The first is the storage system, which handles data movement among the on-chip and off-chip storage. The second component is the compute complex, which comprises one or more processing elements. A processing element can do an image forecasting job independently. By combining numerous processing elements, it uses task-level concurrency. The accelerator operates using a three-stage procedure:

(a) Data configuration: During this phase, the CPU retrieves all necessary data from the outside storage, such as original pictures, kernels, and weights. The information is then sent over Direct Memory Access (DMA) to the storage embedded in the FPGA chip. The centralized controller oversees the flow of data and initiates data processing promptly after information setup.

(b) Data processing: The FPGA acceleration board currently forecasts feed-forward for one or many pictures. The central controller retrieves kernels and loads from storage and saves them in on-chip archives, enabling the processing elements to access the necessary data quickly. The outcomes produced by processing elements are saved in storage.

(c) Result recall: Once all the picture-predicting jobs have been completed, the centralized controller transmits a signal to the CPU, initiating the outcome recall step. The centralized controller retrieves the data from storage and transfers it to the CPU via DMA.

B. Matrix Multiplication Generation

Every MMG comprises a 64-slice line buffering, a 64-slice 3×3 multiplier array, an adder tree, a normalization block, a ReLU block, and a pooling unit (Figure 2). During each convolution, the MMG module retrieves the feature mappings and weighting and stores them in line buffering. The adding tree calculates the sum of the results obtained from multiplying in the multiplier array based on the chosen convolution type.

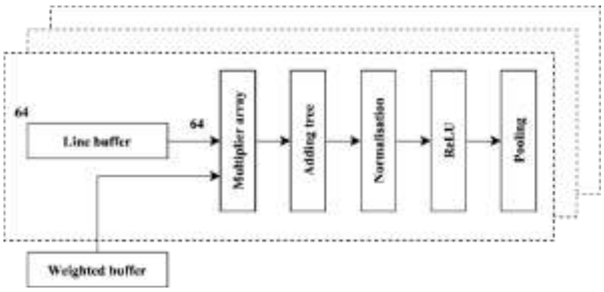

Figure 2. MMG architecture

The following operations are optional normalization, ReLU, and pooling.

1) The control FSM chooses the appropriate length for the line buffering to accommodate various input sizes. The size of the process is equal to (K−1) multiplied by M plus K.

2) Adding tree: The adding tree is configured to perform the addition operation depth wise or point wise. The black lines or sections are common to both forms of convolution. The blue component is used during the process of depth wise convolution. The section functions properly when point wise convolution is used. Any biases are included at this point.

3) Standard convolution: The initial layer convolution utilizes regular convolution to prevent significant data loss. Thus, this accelerator is designed to do the usual convolution with an input mapping channel of 3. The input features mapping always has a channel number of 3.
4) Depth wise convolution: Depth wise convolution applies convolution to every feature mapping independently. The adding tree is designed to concurrently calculate the total results from each section of the multiplying array. The resultant channel number for a single MMG is 32.
5) Point wise convolution: It refers to the regular convolution operation with a kernel size of 1 × 1. The input characteristic mapping is partitioned into multiple M × M × 64 sub-matrices to maximize the benefits of the multiplication in MMG. These sub-matrices are then sequentially transferred into line buffering. This concept derives from the division and conquer process of massive matrix multiplication. The method involves splitting a big matrix into multiple smaller matrices, multiplying each smaller matrix, and then summarizing the results. A single MMG can simultaneously multiplicate M2 × 64 and 64 × 9. Therefore, the resultant channel number is 9.
6) Normalization: Once training is complete, the settings of batch normalization remain unchanged. The intricate process of normalizing is simplified to a combination of multiplication and addition operations.
7) Pooling: Mean pooling and maximum pooling are handled distinctively. By outputting the pixels of a characteristic mapping channel individually, the mean pooling is determined by adding a multiply-accumulate step with a factor of 1/S, where S is the size of the mean pooling. However, max pooling requires an extra step of evaluation.
8) ReLU: Following the normalization phase, a ReLU phase is included, similar to the pooling stage. There are three possible choices: without ReLU, standard ReLU, and ReLU-6.

C.Storage organization

To achieve optimal memory organization, it is necessary to strike a balance between the resources of on-chip storage and outside storage capacity.

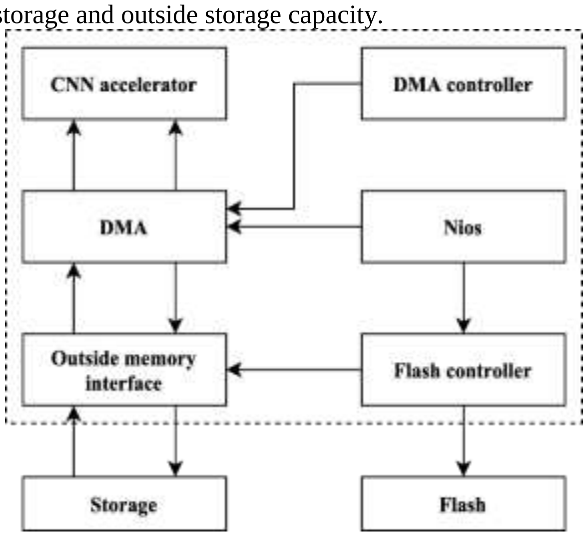

Figure 3. System layout of the FPGA with hardware accelerator

The on-chip storage in FPGA devices has a limited capacity but provides a very high data transfer rate. Outside memory can hold significant information but is restricted in bandwidth. The suggested accelerator uses the hierarchy storage technique. The weight buffering retrieves the necessary variables from external storage before the commencement of every convolution. The system layout of the FPGA with hardware accelerator is shown in Figure 3.
This lowers the delay resulting from loading variables, and on the other hand, it prevents the delay produced by the restricted bandwidth from outside storage. The weight buffer is implemented as a ping-pong buffering. This implies that while weight buffering one outputs information for the current convolution, weight buffering two simultaneously loads data from outside memory for the following convolution, conversely.
Intermediary characteristic mapping is selected to decrease processing duration as part of the system layout. The dimension of the characteristic mapping determines the dimension of the MMG based on the quantity of MMG instances.

4. PERFORMANCE ANALYSIS

This section first presents the process of configuring the setting for the studies. Subsequently, a thorough analysis of experimental findings is presented.

A. Experimental setup

The accelerator layout is executed using Vivado HLS. This tool allows the implementation of the accelerator using the C language and the importation of the RTL as an Intellectual Property (IP) core in Vivado. The C code of CNN architecture has been parallelized by including HLS-defined pragma directives. The parallel variant of the code is then tested using a timing analysis tool. This device's fast pre-synthesis modeling is accomplished by its C simulation and C/RTL co-simulation. Pre-synthesis material reports are used to conduct design area investigations and estimate efficiency.

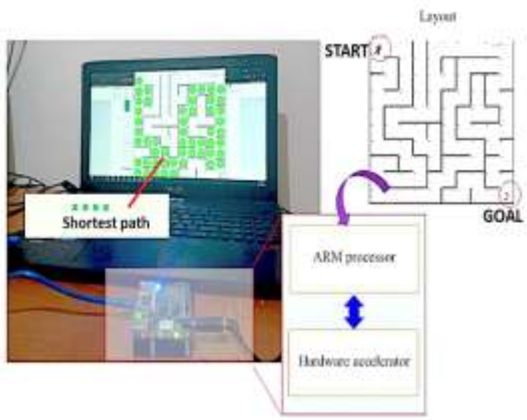

Figure 4. System layout of the FPGA with hardware accelerator

The RTL design is sent abroad, synthesized, and executed using Vivado.

The implementation is based on the board with a Xilinx FPGA chip called Virtex7. The operational frequency of the device is 120 MHz. The software design is executed on an Intel CPU, operating at a frequency of 2.24 GHz and equipped with a cache size of 16 MB. The hardware setup of the research is shown in Figure 4.

B. Experimental results

According to Table 1 the resource usage analysis of CNN-FPGA-HA indicates that the whole system makes high use of FPGA assets, with 97.3% of the Digital Signal Processing (DSP) holdings on the Arria 10 FPGA being consumed. The rationale for this discovery is that optimization aims to increase the processing rate. Therefore, a large number of DSP assets are used for MAC operations. There is a significant excess of on-chip storage space on FPGA, indicating that the suggested network and circuit modeling approach shows potential in managing more considerable map representations to accommodate broader application situations.

Table 1. FPGA resource usage analysis

Resource	Utilization	Available	Percentage (%)
Digital signal processing	1560	1623	97.3
Arithmetic logic module	65050	364260	16.4
Registers	320500	180500	17.5
Random access memory	750600	56850500	14.2

The CNN-FPGA-HA accelerator operates with a system clock that is approximately 150 MHz. This research compares the FPGA-based accelerator with existing software implementations of CNN-FPGA-HA regarding computation time, efficiency, and power consumption. The performance comparison of CPU, GPU, and FPGA are shown in Table 2. The average processing time for 10k photos of size 64×64×3 is evaluated using CNN-FPGA-HA on both the CPU and GPU. Due to the complete pipelining of the design, the computing time for 64×64 pictures is a mere 8.43 microseconds, resulting in a 9-fold increase in performance compared to the GPU system and a 42-fold increase compared to the CPU system.

Table 2. Performance comparison

Model	Processing time (µs)	Speeding up	Power (watts)	Performance (GOP/s)
Central Processing unit	457.2	1.2 X	96.4	7.43
Graphical processing unit	96.54	4.7 X	254.3	38.51
Field programmable gate array	12.34	42.5 X	8.43	328.32

The total MAC calculation for CNN-FPGA-HA is 1.72×112, equivalent to 3.43× 10° activities, as each MAC comprises two calculations. The throughput of CNN-FPGA-HA is determined in GOP/S using the same metric. Based on the trial findings, FPGA acceleration demonstrates a total efficiency of 328.32 GOP/s and offers particular efficiency benefits relative to its software competitors.

Table 3. FPGA hardware accelerators performance analysis

Parameter	Park et al. 2020	Huggi, & Jamuna, 2020	Liu et al. 2021	CNN-FPGA-HA
CNN method	Alex-Net	VGG-16	Alex-Net	Spark-Net
Frequency / MHz	100	150	140	150
DSPs used	1718	810	900	1548
Power (watts)	21.36	9.53	4.26	7.32
CNN dimension (MOP)	1392	30264	483	3.85
Performance (GOP/s)	425.4	127.4	27.4	328.32
Power efficiency (GOP/s/W)	16.45	13.75	6.32	42.31

The CNN-FPGA-HA relies on FPGA, so the prototypes are contrasted with current FPGA-based CNN acceleration. The calculation of the index must be standardized. The cited study elucidates that diverse research organizations employ various FPGA devices and model computing difficulty to attain distinct optimization targets. Performance is the central metric used to quantify the computational power of devices. Electrical usage is the primary aspect of portable edge computation, making the power efficiency index significant. As shown in Table 3, the suggested accelerator achieves a rate of 328.32 GOP/s, surpassing prior FPGA accelerator architectures. Compared with traditional accelerator design, the research's design is considerably improved for energy efficiency.

5. CONCLUSION

This research has effectively shown that using a Virtex-7 evaluation kit to create an FPGA-based acceleration is designed explicitly for CNNs. The Verilog code was carefully designed, demonstrating a solid design that effectively balances computing performance and resource use. In addition, the research integrates row-level pipeline programming and uses a data buffering method to reduce the complications related to data reorganization. The research has created an architectural exploration approach that effectively identifies the best design options for each layer. Based on the experimental findings, the completely pipelined CNN hardware acceleration reaches an output of 328.32 GOP/s with a power consumption of 42.31 GOP/s/w. This indicates that it surpasses the outcomes of prior techniques. This highlights the effectiveness of the accelerator in enabling high-performance CNN computation while preserving significant resources for possible future improvements. The study enhances the existing expertise on FPGA-based CNN accelerators, offering substantial insights for academics and engineers striving for efficient hardware alternatives for deep learning systems.

Acknowledgement
Gomathi N received her ME degree in VLSI Design. She is currently working an Assistant Professor in Erode Sengunthar Engineering College. Also she published two national journals in Bio Medical. Her area of interest includes VLSI Design, Bio medical signal processing.

Dr.Jayasanthi M working as a Professor PSG Institute of Technology and applied Research. Also she published many international journals in VLSI and Biomedical. Her area of interest includes VLSI Design, Bio Medical signal processing and Genetic Algorithm.

References:

Bouguezzi, S., Fredj, H. B., Belabed, T., Valderrama, C., Faiedh, H., & Souani, C. (2021). An efficient FPGA-based convolutional neural network for classification: Ad-MobileNet. *Electronics*, 10(18), 2272.

Curzel, S. (2024). Modern High-Level Synthesis: improving productivity with a multi-level approach. In Special Topics in Information Technology (pp. 15-25). Cham: Springer Nature Switzerland.

Guo, Q., Wu, X. J., Kittler, J., & Feng, Z. (2021). Weak sub-network pruning for strong and efficient neural networks. *Neural networks*, 144, 614-626.

Huggi, S., & Jamuna, S. (2020, July). Design and verification of memory elements using Python. In 2020 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT) (pp. 1-4). IEEE.

Ju, Y., & Gu, J. (2022). A systolic neural CPU processor combining deep learning and general-purpose computing with enhanced data locality and end-to-end performance. *IEEE Journal of Solid-State Circuits*, 58(1), 216-226.

Kang, J. & Kim, T. (2020). PV-MAC: Multiply-and-accumulate unit structure exploiting precision variability in on-device convolutional neural networks. *Integration*, 71, 76-85.

Li, J., & Louri, A. (2021). AdaPrune: An accelerator-aware pruning technique for sustainable CNN accelerators. *IEEE Transactions on Sustainable Computing*, 7(1), 47-60.

Liang, T., Glossner, J., Wang, L., Shi, S., & Zhang, X. (2021). Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461, 370-403.

Liu, S., Fan, H., Ferianc, M., Niu, X., Shi, H., & Luk, W. (2021). Toward full-stack acceleration of deep convolutional neural networks on FPGAs. IEEE Transactions on Neural Networks and Learning Systems, 33(8), 3974-3987.

Park, C., Park, S., & Park, C. S. (2020). Roofline-model-based design space exploration for dataflow techniques of CNN accelerators. *IEEE Access*, 8, 172509-172523.

Petrou, L., Kossifos, K. M., Antoniades, M. A., & Georgiou, J. (2022). The first family of application-specific integrated circuits for programmable and reconfigurable metasurfaces. *Scientific reports*, 12(1), 5826.

Pinaya, W. H. L., Vieira, S., Garcia-Dias, R., & Mechelli, A. (2020). Convolutional neural networks. In Machine learning (pp. 173-191). Academic Press.

Samantaray, S. R., Nanda, S., & Dash, P. K. (2021). A fast and adaptive dynamic phasor estimation algorithm implemented on a field programmable gate array (FPGA). *IEEE Transactions on Industrial Electronics*, 69(2), 2088-2098.

Wang, P., He, X., Chen, Q., Cheng, A., Liu, Q., & Cheng, J. (2020). Unsupervised network quantization via fixed-point factorization. *IEEE Transactions on Neural Networks and Learning Systems*, 32(6), 2706-2720.

Wang, T., Geng, T., Li, A., Jin, X., & Herbordt, M. (2020). FPDeep: Scalable acceleration of CNN training on deeply-pipelined FPGA clusters. *IEEE Transactions on Computers*, 69(8), 1143-1158.

Yang, Y., Huang, Q., Wu, B., Zhang, T., Ma, L., Gambardella, G., ... & Keutzer, K. (2019, February). Synetgy: Algorithm-hardware co-design for convnet accelerators on embedded fpgas. In Proceedings of the 2019 ACM/SIGDA international symposium on field-programmable gate arrays (pp. 23-32).

Zhang, Z., Mahmud, M. P., & Kouzani, A. Z. (2022). Fitnn: A low-resource FPGA-based CNN accelerator for drones. *IEEE Internet of Things Journal*, 9(21), 21357-21369.

Zhu, C., Huang, K., Yang, S., Zhu, Z., Zhang, H., & Shen, H. (2020). An efficient hardware accelerator for structured sparse convolutional neural networks on FPGAs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(9), 1953-1965.

Gomathi N.
Independent Researcher, India
gomathiesec@mail.com
ORCID: 0009-0008-2663-8842

Jayasanthi M.
Independent Researcher, India
ORCID: 0000-1234-5678-9101