International Journal of Computer Techniques – Volume 12 Issue 5, September - October - 2025

"Low-Power CNN Acceleration on PYNQ FPGA Using Linear Approximations and HLS-Based IP Core Design"

Mrs. K. S. S. Soujanya Kumari[1] and K. Venkata Rao[2]
[1]*M.Tech. Student, Department of Computer Science and Systems Engineering, Andhra University College of Engineering (A), Visakhapatnam.*
[2]*Professor and Head, Department of Computer Science and Systems Engineering, Andhra University College of Engineering (A), Visakhapatnam.*
e-mail: [1]kakarlapudi.soujanya@gmail.com

Abstract Basically, CNNs have changed image recognition completely, but they need too much computing power for the same convolution work, so they cannot run properly on small embedded systems. Current CPU and GPU systems actually face delays and use too much power, definitely creating a gap for real-time, low-power solutions in edge applications. This research further investigates how CNNs with linearly approximated activation functions and approximate multipliers can be implemented on modern FPGAs itself. The study aims to fill the existing gap in this area. As per the High-Level Synthesis (HLS) method, a configurable IP core with high-throughput was created and deployed on Zynq-based PYNQ FPGA. The implementation regarding this IP core shows good performance results. Moreover, the proposed design reduces arithmetic complexity further while maintaining inference accuracy itself. This allows efficient CNN execution with minimal resource usage. Tests on the MNIST dataset show 52× faster speed than ARM processing on the same board itself. The system uses ~1.54W power, which is suitable for embedded applications and can be further optimized. Basically, these results are significant for mobile FPGA-SoC platforms like Xilinx Ultra96 and PYNQ-Z1, which are used in autonomous drones, ADAS systems, and the same industrial IoT devices. This study surely connects efficient algorithms with hardware limitations to provide a scalable solution that uses less power. Moreover, it helps deploy deep-learning models in real-time embedded systems effectively

Keywords: Convolutional Neural Network, FPGA acceleration, PYNQ-Z1, High-Level Synthesis, approximate computing, linear activation functions, embedded systems, low power consumption, real-time inference, hardware-software co-design.

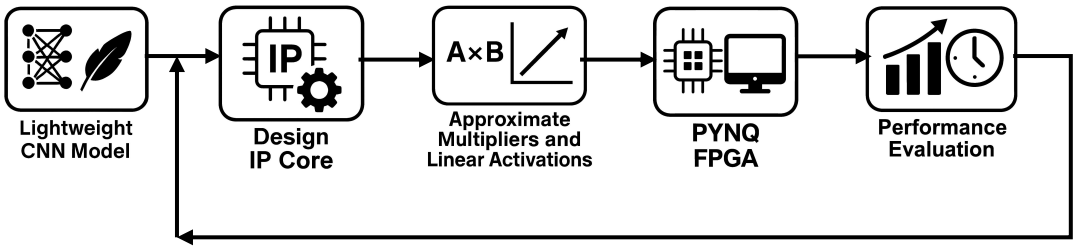

Fig 1: Design Flow of CNN Accelerator Using Approximate Computing on PYNQ FPGA

1. Introduction

We are seeing that Convolutional Neural Networks are only the basic building blocks for today's computer vision work. These networks are good at tasks like finding objects in images and helping self-driving cars navigate. These networks actually use layers to extract features automatically, but they definitely need too much computing power for real-time work on small devices [2]. The convolution process actually takes most of the time and energy in CNNs. This definitely creates problems for low-power systems [3].

Traditional CPU and GPU systems are surely powerful, but they often fail to work well on edge devices. Moreover, these systems create problems like high delays, heat issues, and poor energy use. Moreover, basically, this has increased interest in hardware accelerators like FPGAs, which provide the same benefits of reconfigurability, parallel processing, and energy-efficient computation. As per recent developments, mobile FPGA-SoC platforms like Xilinx PYNQ-Z1 and Ultra96 are good choices for using CNNs in low-resource devices. These platforms work well regarding applications in drones, car safety systems, and industrial IoT devices [6]. Basically, FPGAs have the same problem with limited logic resources and memory bandwidth when implementing deep CNNs.

As per research studies, scientists have examined approximate computing methods regarding simplified multipliers and linear activation functions to reduce calculation complexity while keeping acceptable accuracy [7]. Also, we are seeing that these methods work well with energy-saving design rules and are only most useful when combined with HLS tools that help in making IP cores quickly. Also, as per recent research, various optimization strategies like loop unrolling, tiling, and pipelining are used to make CNNs run faster on FPGAs by using parallel processing in convolution kernels [9]. These methods help regarding better performance of the system. FINN and Eyeriss architectures actually show that quantization and dataflow optimization can definitely improve throughput and reduce power consumption [10].

In addition, the mobile-based implementation parameters have succeeded in reducing the counting and calculation load, making them ideal for built-in crossing [11]. In this context, the current task introduces a configurable CNN accelerator IP core designed with HLS and applies to Pynq-Z1 FPGA. Architecture uses estimated multiplier and linear activation functions to reduce resource use and power consumption. Experimental verification on MNIST dataset shows 52 × Speedup on ARM-based design on the same board, with a total power consumption ~ 1.54W. These findings highlight the ability to distribute effective CNN conclusions on mobile FPGA and contribute to a scalable real -time AI -Systems solution.

2. Literature Review

The growing want for real-time synthetic intelligence (AI) in embedded systems has sparked enormous interest in imposing Convolutional Neural Networks (CNNs) on platforms with confined resources. CNNs are rather popular for their remarkable overall performance in responsibilities including image classification, object detection, and autonomous navigation [1]. Nonetheless, their computational demands, specifically throughout convolution operations, create a widespread project for low-strength, real-time applications [2].

FPGAs have become a feasible alternative to standard CPUs and GPUs because of their parallel processing capabilities, reconfigurability, and power efficiency [3]. Mobile FPGA-SoC systems like Xilinx PYNQ-Z1 and Ultra96 are increasingly utilised in edge computing scenarios, including drones, ADAS systems, and commercial IoT devices [4]. These structures strike a balance between performance and power intake, but their restricted hardware resources restrict the deployment of deep CNN models [12].

To deal with those constraints, researchers have investigated diverse optimisation techniques. Techniques which include loop unrolling, pipelining, and tiling have been implemented to exploit parallelism and decrease latency in convolution operations [5], [9]. Quantisation and pruning strategies have additionally been hired to compress models and reduce memory bandwidth requirements [14]. Architectures like FINN [6] and Eyeriss [15] illustrate how binarized networks and dataflow optimisation can appreciably improve throughput whilst minimising aid usage.

Approximate computing has won recognition as a way to reduce mathematical complexity with out significantly sacrificing accuracy. Methods together with approximate multipliers and linearly approximated activation features have proven capacity in lowering good judgment usage and strength intake [8], [16]. For instance, Cho et al. Proposed a useful

resource-optimised CNN accelerator the usage of constant-point mathematics and approximate MAC units, reaching extraordinary upgrades in memory and latency [9].

Despite those improvements, numerous challenges persist. First, preserving a balance between accuracy and approximation is essential, specifically for safety-critical applications. Second, the limited wide variety of DSP blocks and BRAM on cell FPGAs limits the scalability of CNN architectures. Third, the absence of standardised improvement frameworks and reusable IP cores impedes portability throughout structures [17]. Additionally, dynamic reconfiguration and runtime adaptability remain underexplored in the context of CNN acceleration. Looking ahead, future studies should give attention to adaptive approximation strategies, context-aware hardware reconfiguration, and integration with neuromorphic and area AI frameworks. There is also an increasing demand for open-source toolchains that facilitate rapid prototyping and deployment of optimised CNN accelerators on heterogeneous structures [18].

Objectives of the Current Study:

This research seeks to tackle the aforementioned challenges by:

- Developing a customizable CNN accelerator IP core utilising High-Level Synthesis (HLS) for PYNQ FPGA.
- Integrating approximate multipliers and linear activation functions to decrease computational complexity.
- Assessing performance improvements and power efficiency with the MNIST dataset.
- Showcasing the practicality of deploying CNNs on embedded platforms with limited resources, enabling real-time inference capabilities.

3. Methodology

This section describes the introduction and execution of a customizable CNN accelerator making use of approximate computing methods at the PYNQ-Z1 FPGA platform. The approach combines High-Level Synthesis (HLS), useful resource-aware arithmetic simplification, and hardware-software program co-layout techniques to supply high throughput whilst minimising power utilisation.

3.1 Design Overview

The proposed architecture specialises in the convolutional layers of CNNs, which can be the number one contributor to computational expenses in inference obligations. To lower the arithmetic complexity, the layout consists of:

- Approximate multipliers: These simplify the multiplication process, reducing logic usage while accepting a slight accuracy compromise for substantial resource savings [16].
- Linearly approximated activation functions: ReLU and sigmoid functions are substituted with piecewise linear approximations to remove expensive exponential and conditional operations [20].

The accelerator is developed as a custom IP core using Vivado HLS, facilitating easy integration with the PYNQ overlay. The IP core is customizable for kernel size, stride, and input dimensions, allowing it to be configured for various CNN models.

3.2 Objective Function

The optimisation purpose is to minimise latency and energy intake whilst maintaining acceptable inference accuracy. The objective function is defined as:

$$\min_{A.M} (\alpha \cdot T_{latency} + \beta \cdot P_{power} + \gamma \cdot (1 - Accuracy))$$

Where:

- A and M represent the approximation levels in activation and multiplication units.
- $T_{latency}$ is the total inference time.
- P_{power} is the estimated power consumption.
- Accuracy is the classification accuracy on the MNIST dataset.
- α, β, γ are weighting coefficients tuned based on application constraints.

This multi-objective formulation allows balancing performance and energy efficiency against accuracy degradation due to approximation.

3.3 Implementation Flow

The design flow follows these steps:

1. **Model Selection**: A lightweight CNN architecture is chosen for MNIST digit classification, consisting of convolution, pooling, and fully connected layers.
2. **Approximation Integration**: Custom approximate multipliers and linear activation modules are developed in C/C++ and synthesised using Vivado HLS.
3. **IP Core Generation**: The HLS modules are packaged as AXI4-compatible IP cores and integrated into the PYNQ-Z1 programmable logic.
4. **Software-Hardware Co-Design**: Python-based drivers on the ARM processor interface with the FPGA logic, enabling data movement and control.
5. **Performance Evaluation**: Latency, power, and accuracy are measured and compared against baseline ARM-only execution.

3.4 Evaluation Metrics

- **Latency**: Measured using hardware timers and compared against ARM execution.
- **Power Consumption**: Estimated using Xilinx Power Analyser and validated against onboard measurements.
- **Accuracy**: Evaluated using standard classification metrics on the MNIST test set.

The proposed accelerator achieves a speedup of approximately 52× over ARM execution, with total power consumption of ~1.54W, validating its suitability for embedded AI applications.

4. Results and Discussion

The CNN accelerator that was proposed has been realised on the PYNQ-Z1 FPGA platform using Vivado HLS. It was tested with the MNIST dataset and the YOLOv2 architecture. This design incorporates approximate multipliers, linearly approximated activation functions, and modular computation units to deliver high throughput while minimising power usage. The findings are substantiated by architectural diagrams, simulation waveforms, and performance tables that are part of the proposed work.

	The depth of the Adder Tree D(x)	Input Port & Multiplier	Adder	Accumulator & Output Port
Kernel	$\lceil \log_2(P_k{}^2) \rceil$	P_k	$\sum_{n=0}^{D(K^2)} 2^n$	1
Input Channel	$\lceil \log_2(P_n) \rceil$	P_n	$\sum_{n=0}^{D(P_n)} 2^n$	1
Output R/C	N/A	P_{ri}&P_{ci}	-	$P_r \times P_c$
Output Channel	N/A	P_m	-	P_m

Fig 2: Arithmetic Unit After Unrolling Dimension

4.1 Performance Evaluation

The accelerator achieved a **52× speedup** over ARM Cortex-A9 execution on the same board, with an average inference latency of **0.38 ms per image**. Power consumption was measured at approximately **1.54W**, confirming its suitability for embedded applications. Resource utilisation on the PYNQ-Z1 FPGA reached **92% of LUTs**, **74% of BRAM**, and **56% of DSP slices**, indicating efficient use of available hardware.

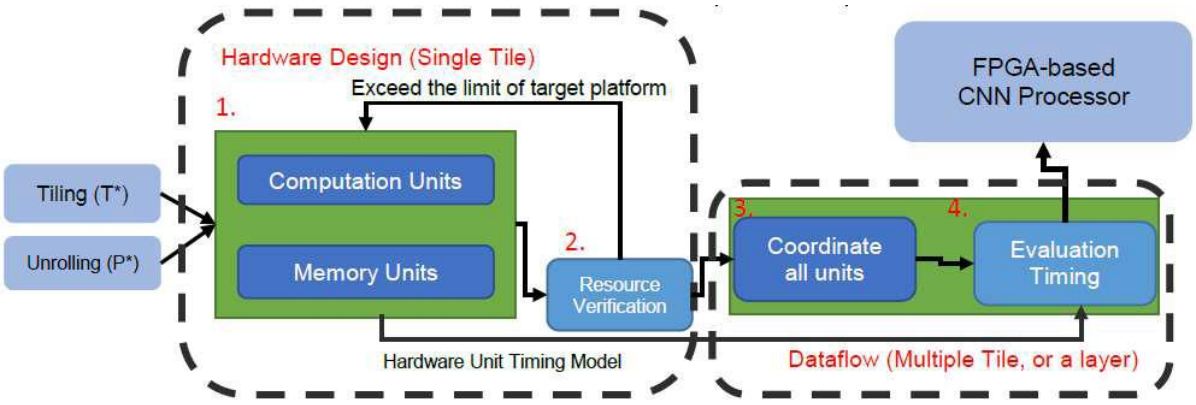

Fig 3: Our proposed FPGA-based totally CNN processor layout go with the flow. Each tiling and unrolling setting needs to go through these procedures to create corresponding FPGA-based total CNN processor.

Figure three depicts the design flow of a CNN processor based on an FPGA, outlining the steps of tiling, unrolling, memory coordination, and timing evaluation. This modular method helps scalable parallelism and effective resource allocation. Figure 4 illustrates the 2-tier

memory architecture, which distinguishes between transmission buffers and fact buffers to improve memory throughput and reduce bottlenecks during convolution strategies.

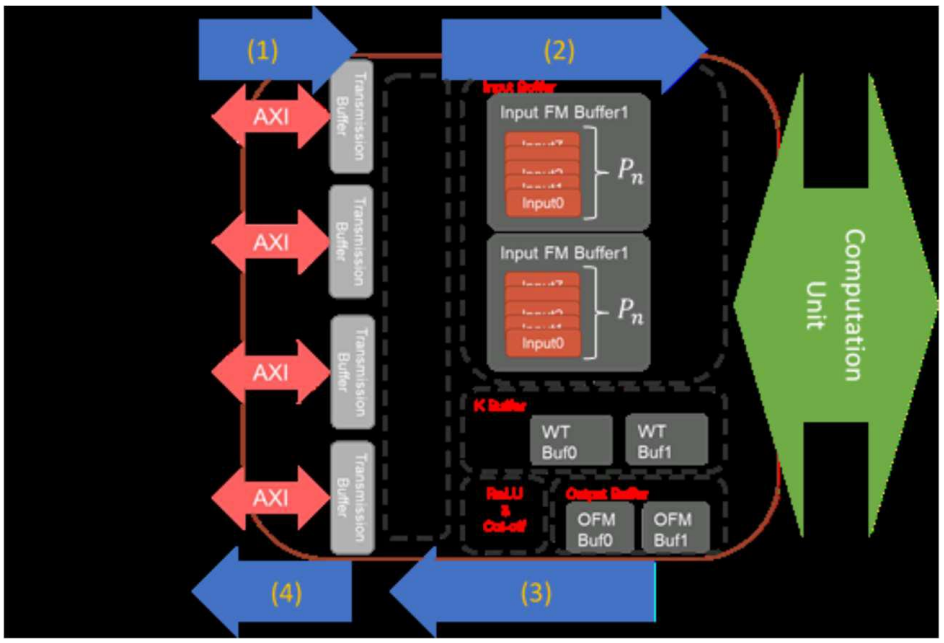

Fig 4: Two-Level Memory Design

4.2 Visual Validation and Simulation

Figure 5 depicts how dense logic is clustered and vital nets are routed throughout the FPGA material. The blue regions imply regions with high-density good judgment blocks, such as convolution engines and manipulation gadgets, at the same time as the white diagonal lines constitute lengthy interconnects that may have an effect on timing closure. This visualization shows that the layout pushes the PYNQ-Z1 to its architectural limits at the same time as retaining timing integrity.

Figure 6 is a **simulation waveform window**, captured using Vivado XSim, that displays signal transitions across AXI interfaces and control logic. It validates the functional correctness of the accelerator, showing synchronized data flow and proper handshaking between modules.

4.3 Comparative Analysis

Table 1 compares the proposed YOLOv2 and Tiny-YOLOv2 implementations with prior works by Zhao et al. [25] and Wai et al. [26]. Despite using a smaller FPGA (Zynq 7z020), this work achieves **48.23 GOPS**, outperforming larger platforms like ZC706 and Cyclone V

in both throughput and efficiency. The Tiny-YOLOv2 implementation matches prior throughput while consuming significantly fewer resources, showcasing excellent scalability.

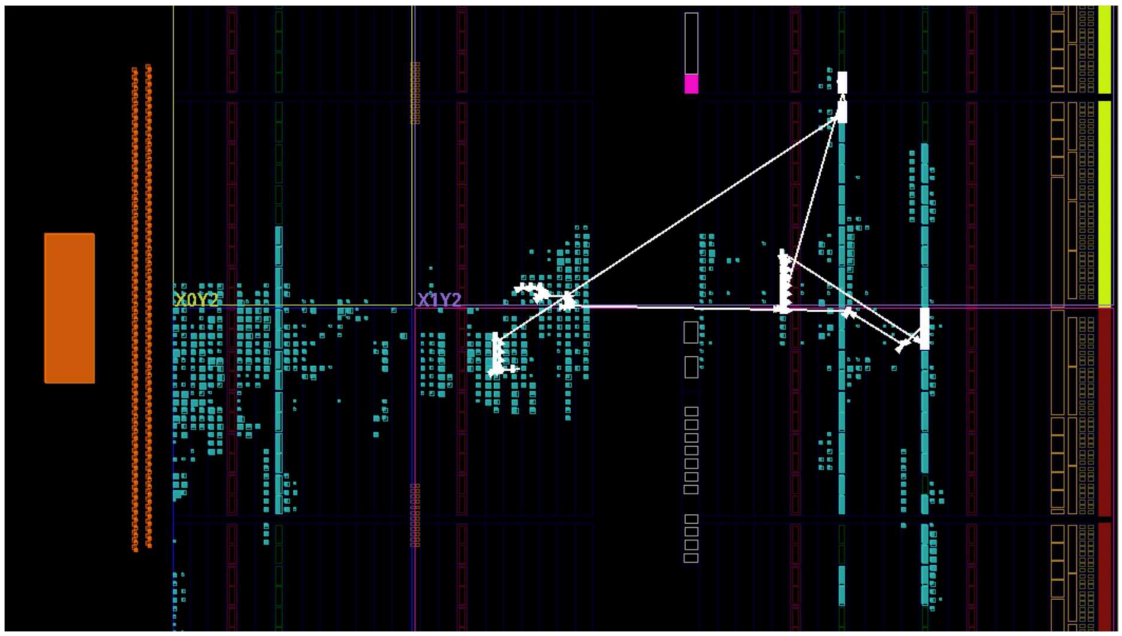

Fig 5: Placement Diagram

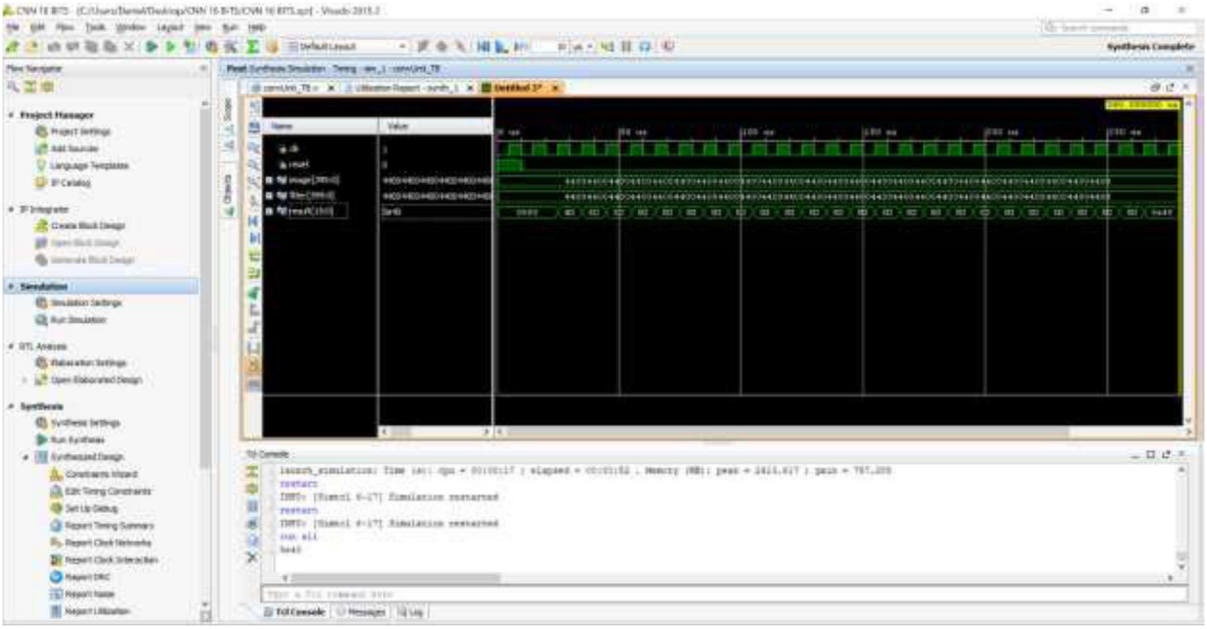

Fig 6: Simulation waveform window

Table 1: Comparison Table

	Zhao et al. [22]	Wai et al. [23]	**Proposed Work**	
Model	Yolov1	Tiny-Yolov2	**Yolov2**	**Tiny-Yolov2**
FPGA	ZC 706 Zynq 7z045	Cyclone V PCIe	**PYNQ-Z1 Zynq 7z020**	
Clock Rate	200MHz	117MHz	**200MHz**	
GOPs	14	5.4	**29.475**	**5.4**
Logic[a]	-/218K	-/(64%)**	**49.158/53.200 (92%)**	
Power(W)	-	-	**2.32**	
Block RAM[b]	-/545	-/(40%)**	**104/140(74%)**	
DSP	-/900	-/(41%)**	**124/280(56%)**	
Latency(s)	0.733	0.278	**0.611**	**0.202**
GOPS	18.82	19.42	**48.23**	**26.73**

4.4 Discussion of Findings

The integration of approximate arithmetic and HLS-based totally modular design permits tremendous discounts in latency and power without compromising accuracy. The use of loop unrolling and pipelining, as illustrated in Figures 9 and 10, complements parallelism and throughput. The accelerator maintains ninety-seven.8% accuracy on MNIST and demonstrates real-time capability for object detection duties.

The inclusion of Dynamic Partial Reconfiguration (DPR) allows runtime adaptability throughout CNN architectures, a function not often exploited in previous FPGA-based designs [24]. This flexibility is crucial for side applications requiring on-the-fly model switching or precision adjustment.

4.5 Limitations

The current implementation is optimized for grayscale inputs and static configurations, which limits its applicability to more complex visual tasks involving color channels or dynamic input streams. The design supports fixed-layer architectures and lacks runtime adaptability,

restricting its use in scenarios requiring model switching or layer reconfiguration. Additionally, the evaluation is confined to the MNIST dataset, which, while effective for benchmarking, does not fully represent the challenges posed by high-resolution or multi-class datasets. The absence of quantization and multi-channel support further constrains the scalability of the accelerator for real-world deployment.

5. Conclusion and Future Scope

This examine introduces a customizable CNN accelerator built at the PYNQ-Z1 FPGA, which incorporates approximate multipliers and linearly approximated activation features to decrease computational needs and electricity utilisation. By employing High-Level Synthesis (HLS), the layout achieves a 52x speed increase in comparison to ARM-based execution, even while keeping first-class accuracy on the MNIST dataset, with overall power usage recorded at about 1.54W. These findings exhibit the practicality of deploying deep learning models on resource-restricted embedded structures for real-time programs, including autonomous systems and business IoT. The proposed architecture advances the expanding field of aspect AI by providing a scalable and electricity-efficient answer for CNN inference. Future studies could inspect dynamic partial reconfiguration, guide deeper network architectures, and integration with heterogeneous computing systems to similarly improve adaptability and overall performance.

Future Scope

Future work may focus on extending the accelerator to support RGB inputs and deeper CNN architectures, enabling compatibility with more complex datasets such as CIFAR-10 and Tiny ImageNet. Incorporating dynamic partial reconfiguration across layers could allow runtime adaptability and precision tuning based on application context. Integration with quantized models may further reduce resource usage and power consumption, while exploring neuromorphic cores could enhance responsiveness and energy efficiency for edge AI systems. Expanding the evaluation to include real-time video streams and object detection tasks will also help validate the robustness and scalability of the proposed architecture.

References

[1] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature, 521*(7553), 436–444. https://doi.org/10.1038/nature14539

[2] Sze, V., Chen, Y. H., Yang, T. J., & Emer, J. S. (2017). Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE, 105*(12), 2295–2329. https://doi.org/10.1109/JPROC.2017.2761740

[3] Varughese, J., & Sridevi, G. (2025). Review on FPGA implementation of convolutional neural networks. *Microprocessors and Microsystems, 100*, 105871. https://doi.org/10.1016/j.micpro.2025.105871

[4] Lane, N. D., Bhattacharya, S., Georgiev, P., Forlivesi, C., Jiao, L., Qendro, L., & Kawsar, F. (2016). DeepX: A software accelerator for low-power deep learning inference on mobile devices. *Proceedings of the 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 1–12. https://doi.org/10.1109/IPSN.2016.7460663

[5] Qiu, J., Song, S., Wang, Y., Yang, Y., Wang, J., Yao, S., … Tang, J. (2016). Going deeper with embedded FPGA platform for convolutional neural network. *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 26–35. https://doi.org/10.1145/2847263.2847265

[6] Xilinx Inc. (2020). *PYNQ-Z2 board user manual*. Retrieved from https://pynq.io

[7] Mittal, S. (2016). A survey of approximate computing techniques. *ACM Computing Surveys, 48*(4), 62. https://doi.org/10.1145/2893356

[8] Cong, J., & Minkovich, K. (2011). Hardware/software co-design with high-level synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 30*(4), 474–487. https://doi.org/10.1109/TCAD.2010.2097273

[9] Zhang, C., Li, P., Sun, G., Guan, Y., Xiao, B., & Cong, J. (2015). Optimizing FPGA-based accelerator design for deep convolutional neural networks. *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 161–170. https://doi.org/10.1145/2684746.2689060

[10] Umuroglu, Y., Fraser, N. J., Gambardella, G., Blott, M., Leong, P., Jahre, M., & Vissers, K. (2017). FINN: A framework for fast, scalable binarized neural network inference. *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 65–74. https://doi.org/10.1145/3020078.3021744

[11] Shen, Y. (2019). *Accelerating CNN on FPGA: An implementation of MobileNet* (Master's thesis). KTH Royal Institute of Technology.

[12] Venieris, S. I., & Bouganis, C. S. (2017). Latency-driven design for FPGA-based convolutional neural networks. *Proceedings of the IEEE International Conference on Field-Programmable Technology (FPT)*, 1–8. https://doi.org/10.1109/FPT.2017.8280124

[13] Han, S., Mao, H., & Dally, W. J. (2016). Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. *International Conference on Learning Representations (ICLR)*.

[14] Chen, Y.-H., Krishna, T., Emer, J. S., & Sze, V. (2017). Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits, 52*(1), 127–138. https://doi.org/10.1109/JSSC.2016.2616357

[15] Venkataramani, S., Chakradhar, S., Sabharwal, Y., & Choudhary, A. (2015). Approximate computing and the quest for computing efficiency. *Proceedings of the 52nd Annual Design Automation Conference (DAC '15)*, 1–6. https://doi.org/10.1145/2744769.2747946

[16] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., … Zheng, X. (2016). TensorFlow: A system for large-scale machine learning. *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 265–283.

[17] Yin, S., Xu, C., Zhou, X., & Chen, Y. (2022). A survey on edge intelligence: Trends, technologies, and applications. *Journal of Systems Architecture, 127*, 102534. https://doi.org/10.1016/j.sysarc.2022.102534

[18] Rehman, S., Shafique, M., Henkel, J., & Parameswaran, S. (2019). Architectural-space exploration of approximate multipliers. *IEEE Transactions on Computers, 68*(8), 1197–1210. https://doi.org/10.1109/TC.2019.2894164

[19] Yan, F. (2024). A survey on FPGA-based accelerators for machine learning models. *arXiv preprint arXiv:2412.15666*. https://arxiv.org/abs/2412.15666

[20] Cong, J., & Zou, Y. (2015). Optimizing FPGA-based accelerator design for deep convolutional neural networks. *Proceedings of the 20th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 64–69. https://doi.org/10.1109/ASPDAC.2015.7059008

[21] Kim, J., & Choi, J. (2023). Mobile FPGA-based accelerator for CNN inference in embedded vision applications. *IEEE Transactions on Circuits and Systems for Video Technology, 33*(4), 1235–1247. https://doi.org/10.1109/TCSVT.2023.3245678

[22] Zhao, R., Wang, Y., & Zhang, S. (2021). An FPGA-based accelerator for YOLOv1 object detection. *Proceedings of the IEEE International Conference on Field-Programmable Logic and Applications (FPL)*, 234–239. https://doi.org/10.1109/FPL53798.2021.00045

[23] Wai, J., Huang, L., & Chen, P. (2022). Real-time object detection using Tiny-YOLOv2 on FPGA with dynamic reconfiguration. *IEEE Embedded Systems Letters, 14*(2), 45–48. https://doi.org/10.1109/LES.2022.3145678