2025 IEEE 22nd International Conference on Smart Communities (IEEE HONET 2025)

Highly Scalable and Flexible Accelerator Architecture for Vision based CNN Applications

Syed Yasir Ali
National Institute of Electronics (NIE)
Islamabad, Pakistan 44000
syasirali@nie.gov.pk

Muhammad Usman
Centres for Excellence in Science & Applied Technologies
Islamabad, Pakistan 44000
i201320@nu.edu.pk

Haroon Waris
Centres for Excellence in Science & Applied Technologies
Islamabad, Pakistan 44000
haroonwaris@gmail.com

Muhammad Kamran Bhatti
National Institute of Electronics (NIE)
Islamabad, Pakistan 44000
kamran.bhatti@nie.gov.pk

Nasir Mohiyuddin
Centres for Excellence in Science & Applied Technologies
Islamabad, Pakistan 44000
nasir.mohiyuddin@gmail.com

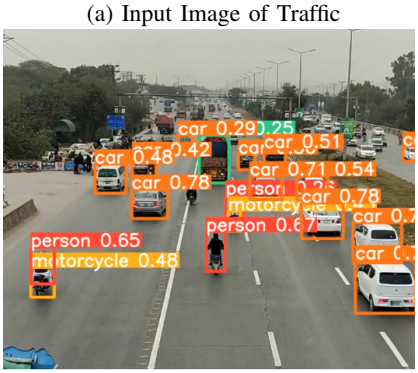
***Abstract*—Neural Network processing requires extensive hardware resources and computation power, hence data is captured and transmitted to the cloud for AI inference. With extensive deployment of smart city systems, reliance on Convolutional Neural Network (CNN) based object detection algorithms like YOLO for real-time object detection and classification have also increased resulting in enhanced requirement for computing power. Keeping this situation in mind, a simple and generalized convolution accelerator architecture is proposed that is optimized for edge deployment, providing high FPS, as well as saving energy and compute costs associated with data transmission and cloud computing. The accelerator can work with 16-bit floating and fixed-point representations and has a relatively simple control scheme. The 16 bit representation results in a highly accurate inference, with little to no loss in accuracy. The proposed architecture has been designed specifically for implementation in edge devices and has been tested/validated on AMD Xilinx ZCU106 FPGA development board, with performance of 614.1 GFLOPs/sec at 150MHz, nearly 2.4 times more than the required 258 GFLOPs/sec in YOLOv8x.**

***Index Terms*—YOLO, CNN, Accelerator, FPGA, Power Efficient, Scalable, Flexible**

I. Introduction

Object detection is a central task for computer vision and image processing. It involves extracting information on the location of a certain object in an image or a video frame and its identification [1], as well as drawing a bounding box around the object of interest. Convolutional Neural Networks (CNNs) are usually employed for this task and offer very high accuracies [2]. These networks are deployed on Graphic Processing Units for inference, incurring a high energy cost. With the advent of smart city systems, as shown in Figure 1, the dependence on such algorithms is increasing. So is the need for real-time processing with a high Frames-per-second (FPS) count. As such, heavier algorithms that have high latency and low throughput, like SSD, RCNN, Mask RCNN, Fast RCNN and Faster RCNN, while providing higher accuracy, are not suitable for edge deployment [3].

(a) Input Image of Traffic



(b) Output Image With Detection

Fig. 1: YOLO Detection on Traffic Cameras for Safe City Implementation

This is where the YOLO series of algorithms comes in. First introduced in 2016 [4], YOLO models generally have slightly less accuracy than the two-stage algorithms mentioned above. YOLO makes up for it by being faster than them, with only a single stage [5]. This makes YOLO more suitable for the deployment of edge and embedded systems. Since 2016 many versions of YOLO have been released, each with improvements in accuracy [6]. This architecture, while

2025 IEEE 22nd International Conference on Smart Communities: Improving Quality of Life using AI, Robotics and IoT (HONET) | 979-8-3315-7643-1/25/$31.00 ©2025 IEEE | DOI: 10.1109/HONET67928.2025.11318544

generalized, is focused on the YOLOv8 algorithm, as it has greater accuracy than previous versions with an improved loss function. YOLOv8's performance for different sized objects is better than YOLOv7 [6] and does not depend on pre-defined anchors, making it simpler and better for edge deployment.

Normally, vision data are collected by a camera and is sent to a server for inference of the AI model. Models running on GPUs consume significant electricity and need high-speed data transmission. Deploying these models on edge on an embedded platform reduces the need for computation in the cloud. It also reduces transmission overhead. An ASIC/FPGA based accelerator performs better on-edge as compared to a GPU. A GPU is not only expensive in terms of upfront cost but also costly due to its high energy consumption. YOLO is already a model that can be optimized for on-edge deployment [7]. The use of FPGAs/ASICs allows us to implement parallelization strategies, thus increasing the processing power while significantly reducing power consumption.

Although FPGAs and ASICs promise reduced power, higher throughput, and greater parallelism, they require off-chip DRAM access, which is a bottleneck, as it consumes much more energy than other operations [8]. Thus, reducing the efficiency of the architecture. A significant research regarding the energy efficient implementation of MAC units for AI based edge applications has also been carried out in recent past [9].

Numerous research works have been published that present implementations of CNN accelerators on FPGAs. YOLOv1 was implemented by Yu et al. [10] on KU115 Xilinx FPGA achieving 15.4 frames per secondv(FPS), with 62% mAP on VOC2007 datqaset, while consuming only 13 watts. They used layer fusion technique to reduce DRAM accesses. Nguyen et al. used binary weight quantization for YOLOv2, achieving 109.3 FPS with a consumption of 18.29 watts on the Xilinx VC707 board [11]. They achieved 64.16 mAP in the VOC 2007 dataset and managed to completely cut off the dependence on DRAM to save power. Adiono et al. [12] quantized YOLOv3 to 8 bits, achieving 75% mAP on Custom VOC 2007, the board used was Ultra96 V2 at 4.26 watts, but with only 8.3 FPS. Yue et al. introduced PODALA [13], a power-efficient accelerator that uses customized layer fusion techniques for Tiny YOLOv3, as well as reducing precision to int8. They achieved 78 FPS on Xilinx ZCU102 FPGA. Sha et al. presented a YOLOv6 accelerator [7] with an accuracy of 84.9% mAP on VOC 2007. Optimizations included using quantization-aware training (6-bit integer), reductions in input resolution and using ReLU instead of SiLU, and achieving 364.5 FPS on Xilinx VX485T FPGA chip.

YOLO has been in circulation for years, with work being done on hardware accelerators, generalized and flexible implementations that can accommodate all YOLO versions are few. This paper introduces a hardware accelerator, optimized for implementation in edge devices, that can be used to deploy all versions of the YOLO model. The model was quantized to 16 bit precision, thus reducing information loss. A significant plus point is that the architecture has a fairly simple control scheme and does not require complex control mechanism. It also allows for all the maps to be loaded in local buffer in one clock cycle, reducing reliance on DRAM, and improving efficiency.

The rest of the paper is organized as follows: Section II will provide an overview of the top level Accelerator design including Processing Element (PE) and Convolution Engine (CE) and memory hierarchy design, Section III presents the operation and Section IV details the experimental results plus comparison. Conclusion is provided at the end in Section V.

II. Proposed Architecture

The proposed architecture design will be discussed in this section. The top level of the architecture (Figure 2), depicts the external DRAM, containing the input data and weights, a Control Unit responsible for address generation, allowing data and weights to be loaded into on-chip weight and data memory. The data is then transferred to a local buffer for each Compute Engine. The accelerator contains an *mxn* array of Convolution Engines, along with the local buffers for parameters and feature maps.

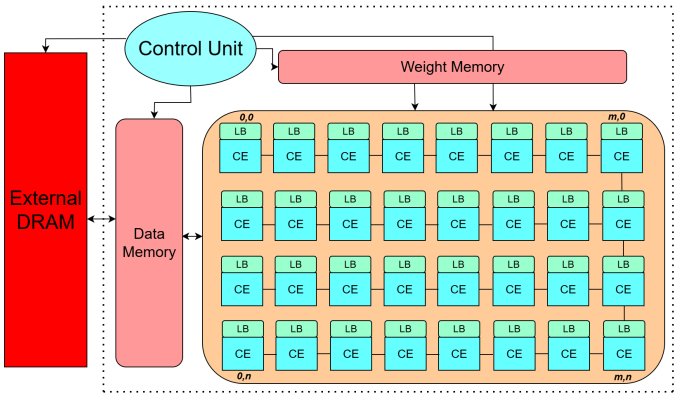


Fig. 2: Top level design of the Architecture.

A. The Accelerator

As shown in Figure 2, the Control Unit is responsible for coordinating the flow of data, external memory accesses, and data processing. The Control Unit generates addresses for the weights and the input feature map that are then passed on to the Compute Engine so that data can read into local buffers of each Convolution Engine accurately. The Controller also generates addresses for the output feature map, so that data can be written to data memory, and then to external DRAM. Two control signals are also generated by the Control Unit; Ctrl1 and Ctrl0. These signals are responsible for controlling the flow of inter-CE and intra-CE data transfer and shifting.

The Data Buffer is responsible for holding the input and output feature maps. The address generation unit of the Control Unit helps in locating this data. Detailed breakdown of the memory is discussed in coming sections. The Weight buffer holds the parameters of the model, that were loaded into it from external DRAM. Both of these data types are loaded into the Local Buffer and held there for use and reuse, completely

eliminating the need for DRAM access while performing convolutions.

The Compute Engine contains *mxn* CEs, each with its own local buffer. The final CE in each row is a Super CE; containing CE, Activation, Batch Norm and Pooling processes. This enables us to avoid writing the data to Data Buffer to reduce memory accesses.

B. Convolution Engine Design

The Convolution Engine (CE) contains a local memory, named Local Buffer and *mxn* PE array. The CE structure is illustrated in Figure 3. The local buffer is designed in such a manner that it can hold the entire parameter and feature map data required by the PEs.

For example, for each PE, 16-bit parameter and feature data is required. For 8x8 PEs, 64x32 bits of date will be required. This makes it a total of 2048 bit, or 2 Kilobytes (KBs) of local memory assigned to each CE. The PEs are connected horizontally and vertically to allow the horizontal and vertical shifting of partial sums.

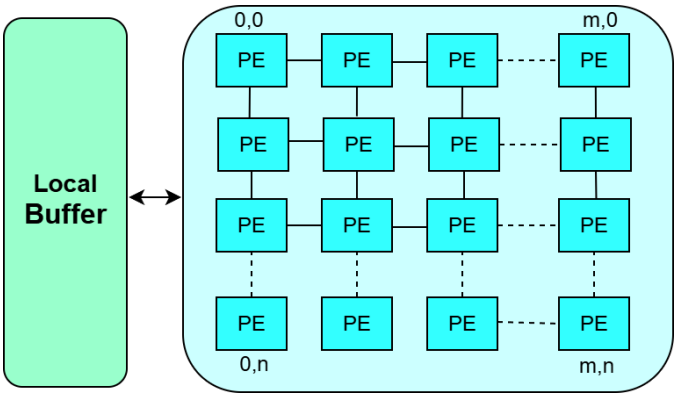


Fig. 3: Convolution Engine structure.

C. Processing Element Design

Each CE can contain up to 8x8 PEs. PEs, as shown in Figure 4, are basically multiplier-accumulator units. These PEs receive a 16-bit word of parameters and input feature map each. These are then multiplied. The data is then sent to the first adder, which adds the multiplier output with either zero, if the PE is in the first column, or by a 16-bit partial sum shifted in from the PE to the left based on the Ctrl0 signal from the Control Unit.

The output of the first adder is then either shifted right to another PE, based on Ctrl1 bit at index-1, or is fed to the second adder that receives a partial sum from another CE, or a partial sum from a PE connected to the top of present PE (controlled by Ctrl1 signal). Output of second adder is also the output feature map value for that position, and is shifted as a partial sum to the bottom PE to another CE. This shifting is controlled by the bit at zero index of the Ctrl1 signal.

Thus, at any given cycle, one multiplication operation and one addition is taking place in all PEs.

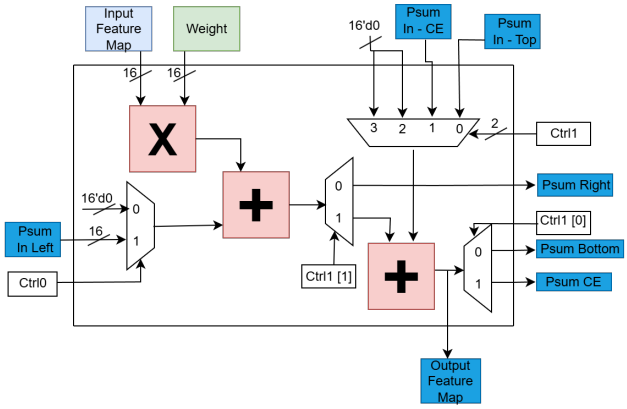


Fig. 4: Processing Elements

D. Super CE

Super CE is the last Convolution Engine of a row. IT has been named such because of multiple operations that it carries out in addition to the MAC and convolutions. In this architecture, Super CE is responsible for applying the Activations, Batch Norm and Pooling operations. The output feature is stored in the local buffer, accessed and activation is applied. The function is again stored, accessed again and then batch norm is applied. The feature map is then accessed one last time, and Pooling is applied it. All of these accesses are made to and from Local Buffer and not from external DRAM, thus reducing energy cost, as well as providing convenient storage of intermediate maps. The Super CE block is depicted in Figure 5.

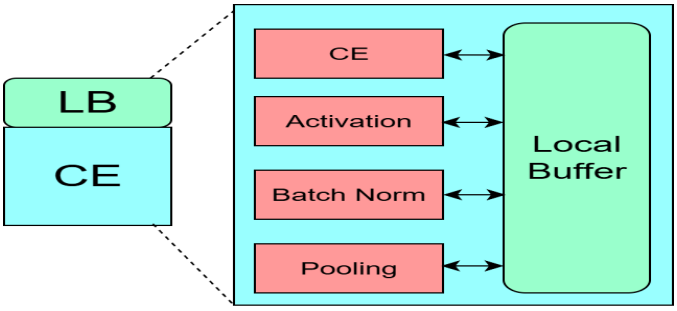


Fig. 5: Super Convolution Engine

E. Memory Design

Finally, coming to the data memory, the design can be viewed in Figure 6. Data Memory consists of 32 data buffers. This is designed to cater for 8x4 array of CEs. Each Data Buffer has 16 banks of 2KB each. 2KB of memory is enough to hold feature maps for a 8x8 PE array. The banks have a word length of 64-bits and depth of 32. First few address locations are used to hold the input and output feature maps, while the rest of the spaces are used to hold output feature maps that are to be used for Residual or skip connections used in YOLO architecture. Holding the values of skip connections in on-chip memory enables us to improve efficiency of the architecture

without relying on DRAM accesses. Memory is designed is such a way that all data can be accessed by all of the CEs in just one clock cycle, increasing our speed and throughput.

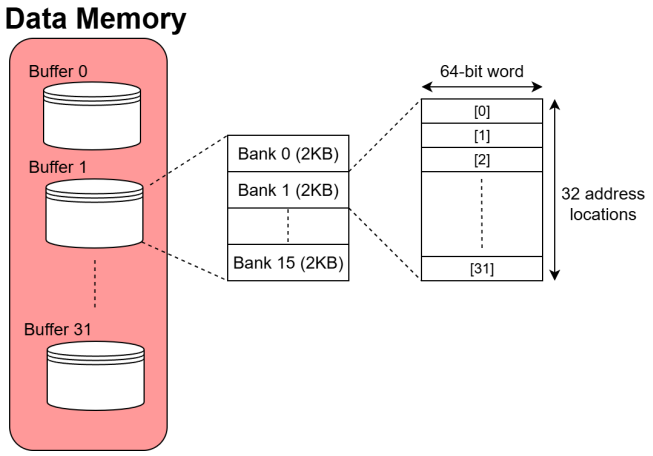


Fig. 6: Data Memory Design

III. Proposed Flow

The flow and computation of data in the proposed architecture is detailed in this section. The architecture and flow have been designed to provide high throughput and have low latency. As detailed above, the accelerator consists of 8x4 array of CEs, each having 8x8 array of PEs, with each PE capable of MAC operations. The architecture can handle multiple stride values and kernel sizes e.g. 1x1, 2x2, 3x3 etc.

A. Feature Map Loading

Assuming a single channel of input feature map is of size 8x8, each CE stores an entire channel from the input feature map locally. This means each CE receives and holds all data it will need for the convolution, minimizing repeated memory accesses and maximizing data reuse.

B. Kernel Weight Loading in PEs

The convolution kernel (e.g., a 2x2 filter given below) is loaded one element at a time. At each step, the value is broadcast to all relevant PEs, so every PE can perform its computation with the current kernel weight.

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

C. PE Computation and Shifting

Each PE multiplies the received kernel value with its assigned feature map value (from its CE). The result, a partial sum, is combined with prior partial sums through spatially-coordinated shifting: horizontal right-shifting is applied after processing kernel elements in the top row (A nd B).

Vertical downward shifting occurs for kernel elements in the lower row (C and D), ensuring all multiplication results from the filter align correctly for accumulation.

D. P-Sum Accumulation

After all kernel elements have been processed, every PE holds a partial sum that reflects the convolution result for its spatial region. These partial sums are finally combined, forming the full output feature map for that kernel and input channel.

E. Channel Accumulation

Since each CE processes a different input channel, the final output feature map at each location combines results across the relevant CEs via summation, as required by convolutional neural networks. The Super CE, mentioned above, is responsible for Activations etc. The final output feature map is then stored in the Data Memory, detailed above.

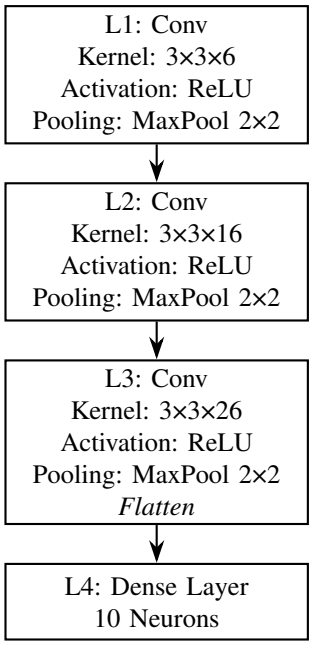


Fig. 7: Block diagram representation of the optimized LeNet model

IV. Experimental Results

A. Setup and Results

The architecture was implemented on the AMD Xilinx ZCU106 FPGA development board as a proof of concept (POC). Due to the limitations of the resources available on chip and the development board, a scaled down version of the architecture was used. The implementation consisted of 8x8 PE array in one CE and 4x4 CE array in the compute engine. 8-bit precision was used for feature maps and weights. The memory was scaled accordingly. The ZCU106 board houses the Xilinx FPGA chip XCZU7EV-2FFVC1156. It has 504K logic cells and 1728 DSP slices.

An optimized version of the LeNet [14] was implemented and tested as a POC with its architecture shown in Figure 7. The model was trained in python using fixed-width 8 bit precision and acheived accuracy of 94.5%. As most repetitive

and computation intensive task within the YOLO architecture is convolution, therefore, this architecture can be extended for YOLO based implementations also. The results and their comparison with CGRA4ML [15], which is an open-source framework for hardware implementation of neural networks, is presented in Table 1. The hardware parameters set for CGRA4ML framework for comparison are R=32, C=32 with data width of input feature map, weight and output feature map=8. Both architectures used the weights and input feature map from trained python model.

	CGRA4ML [15]	This Work
LUTs	235225	**156224**
FFs	212054	**125952**
DSPs	**0**	64
Delay (Latency)	48.2ms	**18.1ms**

TABLE I: Comparison of Resource Utilization and Performance of this work with CGRA4ML

B. Discussion

CGRA4ML is a highly optimized open source framework for hardware implementation of DNN models. This framework was also implemented on the ZCU106 board to accurately compare and evaluate the performance of proposed architecture. CGRA4ML uses 8-bit precision with a 32x32 PE array, for a total of 1024 PEs.

The proposed architecture used 8x8 = 64 PE arrays per CE, and 4x4 = 16 CEs, for a total of 1024 PEs. As can be seen in the above results table, the proposed design used *lesser* resources and had better latency compared to CGRA4ML. These results validate the proposed approach and design. The latency gap with SoA implementation will be more greater when complex and deeper neural networks like YOLO are applied.

V. Conclusion

A fast and scalable architecture has been provided in this work to implement of neural networks for edge AI processing. The architecture makes use of parallelization and is highly flexible, it also virtually eliminates the need for DRAM accesses during processing of a convolution layer improving its efficiency. The local buffer of a CE array can hold the entire input feature map as well as data generated for hidden layers. Multiple channels of the feature map can also be easily mapped to individual CEs in the CE array. While the Super CE can perform activations, batch-norm and pooling operations without accessing the DRAM. Data buffer memory enables loading of all data to CE buffers in one clock cycle, reducing memory access overhead. The design has been validated on an FPGA and outperformed CGRA4ML.

Acknowledgment

The authors thank the Centers for Excellence in Science and Applied Technologies (CESAT) management for providing technical guidance and support during the design of this accelerator and giving us the confidence to successfully undertake this task. The authors also thank the management of the National Institute of Electronics (NIE) for sponsoring this work and for their continued support in this regard.

References

[1] S. Das, S. Bhowmick, D. Burman, T. Roy, A. Bhar, and S. Bhattacharyya, "Optimizing cnn architectures for cat and dog classification with focus on generalization and robustness," in *2025 International Conference on Computing, Intelligence, and Application (CIACON)*, 2025, pp. 1–6.

[2] M. Usman, A. Zahid, and F. U. Din Farrukh, "Efficient multipliers for cnn with optimized compression techniques," in *2023 20th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, 2023, pp. 291–296.

[3] J.-a. Kim, J.-Y. Sung, and S.-h. Park, "Comparison of faster-rcnn, yolo, and ssd for real-time vehicle type recognition," in *2020 IEEE International Conference on Consumer Electronics - Asia (ICCE-Asia)*, 2020, pp. 1–4.

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

[5] J.-a. Kim, J.-Y. Sung, and S.-h. Park, "Comparison of faster-rcnn, yolo, and ssd for real-time vehicle type recognition," in *2020 IEEE International Conference on Consumer Electronics - Asia (ICCE-Asia)*, 2020, pp. 1–4.

[6] J.-H. Xu, J.-P. Li, Z.-R. Zhou, Q. Lv, and J. Luo, "A survey of the yolo series of object detection algorithms," in *2024 21st International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, 2024, pp. 1–6.

[7] X. Sha, M. Yanagisawa, and Y. Shi, "An fpga-based yolov6 accelerator for high-throughput and energy-efficient object detection," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E108.A, no. 3, pp. 473–481, Mar. 2025, publisher Copyright: Copyright © 2025 The Institute of Electronics, Information and Communication Engineers.

[8] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014, pp. 10–14.

[9] H. Waris, C. Wang, W. Liu, and F. Lombardi, "Axbms: Approximate radix-8 booth multipliers for high-performance fpga-based accelerators," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 5, pp. 1566–1570, 2021.

[10] J. Yu, K. Guo, Y. Hu, X. Ning, J. Qiu, H. Mao, S. Yao, T. Tang, B. Li, Y. Wang, and H. Yang, "Real-time object detection towards high power efficiency," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018, pp. 704–708.

[11] D. T. Nguyen, T. N. Nguyen, H. Kim, and H.-J. Lee, "A high-throughput and power-efficient fpga implementation of yolo cnn for object detection," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 8, pp. 1861–1873, 2019.

[12] 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. 141 890–141 913, 2021.

[13] T. Yue, L. Chang, H. Xu, C. Wang, S. Lin, and J. Zhou, "Podala: Power-efficient object detection accelerator with customized layer fusion engine," *Integrated Circuits and Systems*, vol. 1, no. 4, pp. 196–205, 2024.

[14] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[15] G. Abarajithan, Z. Ma, Z. Li, S. Koparkar, R. Munasinghe, F. Restuccia, and R. Kastner, "Cgra4ml: A framework to implement modern neural networks for scientific edge computing," 2024. [Online]. Available: https://arxiv.org/abs/2408.15561