International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

A Low-Latency, Area-Efficient Convolution Network for FPGA Acceleration

[1]Gutti Naga Swetha, [2]Dr. Anuradha M. Sandi

Submitted: 29/01/2024 **Revised**: 07/03/2024 **Accepted**: 15/03/2024

Abstract: The goal of the technology known as Field Programmable Gate Arrays (FPGA) is to improve the safety, performance, and efficiency of cryptographic operations in contexts with limited resources. The use of deep learning has been more important in recent years, particularly with regard to the achievement of low latency and space efficiency in FPGA-based implementations. This study paper gives According to the suggested model, which is called EffiConvNet (Efficient Convolution Network), ternary neural networks, logic expansion, and block convolution are all integrated. Block Convolution is a technique that tries to optimise the data dependence among the spatial tiles. This helps to ease the load on chip memory and facilitates efficient processing. In order to do this, logic expansion is used, which replaces the XNOR gates with neural networks. This allows for more effective utilisation of resources. In order to achieve the desired degree of efficiency at the training stage, further ternary neural networks are used. The experimental results of our technique on real-world tasks reveal that it is successful. Furthermore, these coupled architectures together (EffiConvNet) illustrate the efficacy of our approach. While assuring optimal resource utilisation and better inference performance, the combination strategy that has been described offers a potential option for addressing the obstacles that are connected with the deployment of large-scale neural networks on FPGAs.

Keywords: *EffiConvNet, Area Efficient, Low Latency, FPGA*

1. Introduction

In today's linked world, secure communication is essential. Cryptography protects data transfer by ensuring secrecy, integrity, authentication, and non-repudiation. Symmetric key cryptography has long been used, but key management and key sharing security concerns plague it.In response, Elliptic Curve Cryptography (ECC) is a potential public-key cryptosystem alternative. ECC uses the algebraic structure of elliptic curves over finite fields for several benefits. These include smaller key sizes, faster calculations, and comparable security to RSA. Such qualities make ECC ideal for resource-constrained contexts like wireless mobile communication and personal communication systems.However, implementing ECC securely and efficiently is complex. Optimised algorithms and structures for ECC arithmetic operations like point multiplication and division are needed. ECC deployment on hardware platforms like Field Programmable Gate Arrays (FPGAs) has issues related to space utilisation, processing latency, and energy efficiency [1]-[3]. Field Programmable Gate Arrays (FPGAs) are commonly accessible programmable devices that enable hardware customisation with low development costs. These devices feature customisable logic blocks (CLBs) with programmable logic cells and a flexible connector network. Programmable input/output cells surround the core. FPGAs also have DSP blocks for arithmetic applications like multiply-and-accumulate. Block RAMs, look-up tables, flip-flops, clock control units, high-speed I/O interfaces, and more are included. This study addresses the complex issues of safe and effective cryptographic implementations in wireless mobile and personal communication systems. Field Programmable Gate Arrays (FPGAs) are used to optimise Elliptic Curve Cryptography (ECC) algorithms and hardware designs. Here are the ways in which deep learning contributes to these objectives:

- **Model Compression:** Deep learning techniques, such as pruning, quantization, and network architecture design, enable the compression of neural network models. By reducing the model size, fewer resources are required to implement the network on an FPGA, resulting in improved area efficiency. Smaller models also lead to reduced memory requirements and lower data transfer, which contributes to lower latency.
- **Optimization for Parallelism:** Deep learning models can be optimized to leverage parallelism, which is essential for efficient FPGA implementations. Techniques like model parallelism and layer parallelism divide the network into smaller subcomponents that can be processed in parallel on FPGA resources. By distributing the computations across multiple processing elements, the overall latency is reduced, and FPGA resources are utilized more efficiently.

[1]*ECE department, Guru Nanak Dev Engineering College,VTU, Bidar, Karnataka, India*
Email - nswethag@gmail.com
[2]*ECE department, Guru Nanak Dev Engineering College, VTU, Bidar, Karnataka, India*
Email - anu29975@gmail.com

- **Hardware-Aware Model Design:** Deep learning models can be designed with hardware constraints in mind, allowing for better utilization of FPGA resources and reducing area overhead. Techniques such as custom layer design, precision optimization, and dataflow optimizations enable the mapping of specific operations to FPGA primitives and take advantage of the hardware's parallelism and computation capabilities.
- **Quantization and Fixed-Point Arithmetic:** Deep learning models can be quantized to lower precision, such as using fixed-point arithmetic instead of floating-point operations. This reduces resource requirements, as fixed-point arithmetic consumes fewer FPGA resources compared to floating-point arithmetic. Quantization also reduces memory bandwidth requirements, leading to lower latency by minimizing data transfer.
- **Model Pruning and Sparsity:** Deep learning techniques like model pruning and sparsity regularization help reduce the number of parameters and activations in the network. Sparse representations enable more efficient storage and computation, resulting in improved area efficiency and lower latency.
- **Hardware-Accelerated Inference:** FPGA platforms offer the flexibility to design customized hardware accelerators for deep learning inference. By implementing key operations of the neural network directly in hardware, the latency is significantly reduced compared to software-based implementations. Deep learning frameworks provide tools to deploy models on FPGAs and generate optimized hardware designs.

By leveraging deep learning techniques and optimizing models for FPGA implementations, it is possible to achieve area efficiency and low latency. Deep learning models, especially Convolutional Neural Networks (CNNs), have demonstrated unparalleled performance in various tasks but necessitate substantial computational resources. When mapped onto FPGA platforms, which inherently possess resource limitations, the tension between optimizing for area and latency becomes particularly pronounced. Current studies often emphasize either minimizing resource utilization or reducing latency, yet rarely combine both ambitions into a cohesive architecture. Consequently, the potential for comprehensive solutions that address both area efficiency and low latency remains largely untapped. Despite the progress in FPGA-based DL acceleration, there remains an evident research gap in the concurrent optimization of area efficiency and low latency. Existing studies often prioritize one objective over the other, leading to suboptimal solutions. This gap stems from the lack of a comprehensive approach that holistically addresses both challenges in FPGA-accelerated DL systems.

1.1 Motivation and contribution

In the evolving realm of artificial intelligence, deep learning has cemented its place at the forefront, particularly in applications necessitating robust image processing capabilities. FPGA (Field-Programmable Gate Array) platforms, renowned for their adaptability, present a unique opportunity for the deployment of deep learning models. The research contribution is as follows:

- our research contributions encompass novel techniques for improving FPGA-based DNN acceleration. By introducing block convolution, logic expansion, and integrating TNNs,
- we offer a holistic approach that advances the field's understanding of efficient neural network deployment on FPGA platforms.
- Our work contributes to addressing memory limitations, enhancing resource utilization, and optimizing training processes for improved area efficiency and low-latency DNN inference.

This research is organized as follows: first section starts with background of FPGA and integration of deep learning to optimize its performance. Furthermore, motivation and contribution is discussed. Second section discusses brief review of existing relevan work, third section presents the mathematical modelling of proposed model. Proposed model is evaluated in fourth section along with comparative analysis with existing model.

1.2 Objectives

In the field of deep learning, the implementation of large-scale neural networks on FPGAs faces challenges in achieving area efficiency and managing latency constraints. The proposed EffiConvNet model seeks to address these challenges by integrating innovative concepts such as ternary neural networks, logic expansion, and block convolution. The overarching problem is the need for a cohesive approach that harmoniously combines the optimization of cryptographic operations and deep learning within FPGA platforms, aiming to strike a balance between security, efficiency, and computational performance.

1. Develop arithmetic operations in ECC Core.

- Point Multiplication Architecture.
- Point division architecture.

2. Implementing the architecture efficiently on FPGA in terms of area.

3. Architectural design focused on achieving reduced latency.

4. Energy-efficient design for Field-Programmable Gate Arrays (FPGA).

2. Related Work

Enhancing the efficiency of compressed Binarized Neural Network (BNN) models involves employing pruning methods to eliminate redundant parameters. However, there exists a trade-off between accuracy and pruning, as higher pruning rates may lead to decreased accuracy. In [7], the authors leverage Bayesian optimization for channel pruning in quantized neural networks. This approach is based on preserving the angles of high-dimensional binary vectors [8] and considering euclidean distance. In a similar vein, [9] introduces neuron pruning for fully connected layers, followed by network retraining. On the other hand, [10] presents a learning-based technique to prune the number of filters/channels in BNN.The AutoPrune approach proposed in [11] adopts a gradient-based search to optimize a group of learnable parameters, providing an alternative to directly pruning weights. Additionally, [12] employs a weight flipping frequency method to prune BNN, analyzing binary weight sensitivity to accuracy. Moreover, this framework supports layer-wise pruning, reducing the number of channels in each layer by a consistent percentage of insensitive weights. In the realm of BNN, [13] introduces O3BNN-R, which employs two irregular pruning methods for eliminating redundant edges during inference: threshold edge pruning and pooling edge pruning. [14] explores the concept of reusing calculated partial outputs of duplicated filters to prune redundant operations in BNN. The Slimming Binarized Neural Network (SBNN) from [15] utilizes two compression techniques: filter pruning and knowledge distillation.A different perspective comes from [16], which proposes floating-point (FP) feature map compression for a hardware accelerator. This involves hardware design and a compression algorithm, which can accommodate quantization methods like ternary neural quantization (TTQ) without significant accuracy degradation, reducing computational costs. In the hardware architecture domain, [17] presents FantastIC4, an innovative design that supports efficient on-chip execution of multiple compact fully-connected layer representations. The architecture minimizes required multipliers for inference, introducing robustness to 4-bit quantization and high compressibility through a novel entropy-constrained training method.[18] employs algorithm-architecture-circuit design optimization, inspired by data reuse and sparsity in Deep Belief Network (DBN) learning algorithms. This leads to a heterogeneous multicore architecture with localized learning capabilities. Addressing streaming applications, [19] proposes a tailored streaming hardware architecture for improved compute efficiency in CNNs on FPGAs. The accelerator unifies computational functions like convolutional and deconvolutional layers, optimizing residual and concatenative connections to support various CNN topologies during inference.While [20] offers qualitative analysis, [21] provides more detailed insights with quantitative data on inference accuracy, latency, throughput, power consumption, and efficiency. However, both studies suffer from the limitation of using different DNN models for distinct automation frameworks, making direct comparison challenging.

3. Proposed Methodology

Machine learning technologies like neural network setup and Tiled Convolution improve programmable logic devices' area efficiency and latency. These flexible hardware components may be modified to improve neural network accuracy and performance. Programmable logic device setup optimises memory use, including Truth tables. Convolutional neural networks and Tiled Convolution enhance device latency. Feature mappings require improvised latency, causing Convolutional Neural Network (CNN) overhead on Programmable logic devices with limited memory. These methods reduce memory restrictions and improve efficiency for wide-scale processing of Convolutional Neural Networks on Programmable Logic Devices and optimise their topologies. This optimises CNN's memory and programmable logic device implementation, making it quicker and more efficient on a large scale. These devices are thus useful for healthcare imaging, image recognition, and other Convolutional Neural Network applications. These methods optimise CNN's Programmable logic device implementation and memory efficiency. Tiled convolution breaks the input picture into tiles, reducing memory use and processing requirements for large-scale Convolutional Neural Networks. The PLD is optimised for Convolutional Neural Network processing. Machine learning is used to train Convolutional Neural Network data models and programmable logic device designs. This helps the model discover optimum setups. Split tiles interpret the information during tiled convolution. Truth tables and programmable logic device neural network setup improve device resource allocation. The gadget optimises tiny input picture tiles. After this, the Convolutional Neural Network is built on the Programmable Logic Device, which improves its performance due to memory utilisation and device settings.

1.2.1 Architecture and Working of Proposed Model

The detailed view of the proposed architecture is given in the figure 1 below. An input image is taken as an inference. This image is then split up into tiles by the Tile convolution that allows parallel processing and decreased memory utilization therefore increasing the latency of the working model. Every tile is processed separately for efficient use of memory on the Programmable Logic Device. These resulting tiles are then passed on to the 'Learning Programmable logic device configuration of neural networks', at this stage machine learning algorithms are used for optimal configuration of Programmable Logic

Devices. At this stage, the device is both enhanced in latency and has improved area efficiency due to the techniques applied. The fragments or tiles of the image is then concatenated to form the output image.

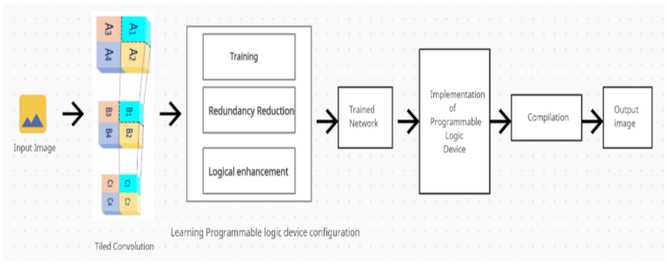

Fig 1 EffiConvNet working module

The tiled convolution uses a ‘divide and merge computational approach’, where the information of the tiles is independent to the output of adjacent tile information. For instance, consider the input image has three features and convolutional layers are 3 by 3 by 3. The input image is split into four tiles as 4 by 4 by 3. The input feature image is given as (X_{input}, I_{input}) whereas the dimension of the output image is given as:

$$I_{output} = 1 + ((I_{input} - l + 2q)t^{-1})$$

$$X_{output} = 1 + ((X_{input} - l + 2q)t^{-1}) \qquad (1)$$

The output image is of the dimension 8 by 8. Considering the tiled convolution proposed in this study, the input image is tiled to the size 5 by 5 that results in an output image tile of 3 by 3. This is then combined to produce an output feature image of size 6 by 6, keeping in mind the size of the input image. This is done by use the Border Extension technique, the method involves added additional pixels around the ends of the image for tiled computations. This assures the size of the input feature image is suitable to the required tile dimension for processing. This also prevents loss of data and information. For instance, an input image tile having original dimension of 5 by 5 is bordered to 6 by 6 pixels where the original feature image is 4 by 4 pixels. This produces output image tiles of dimension 4 by 4 that are combined to produce a resulting output image of size 8 by 8.

$$O = \frac{\left((t^{-1})(J + 2q - l)\right) + 1}{\left(\left(\frac{J}{O} + 2q_u - l\right)(t^{-1})\right) + 1} \qquad (2)$$

In the above equation (2), input feature images are denoted as J for the dimension (X_{input}, I_{input}). Tiled convolution breaks up the input image into smaller fragments. After which Tiled Convolution is performed on every fragment separately. Further the results are concatenated together. Considering the Convolutional Neural Network used for this study, the border extension is performed for CNN multilayers in two different ways, namely ‘Fixed Bordering’ that splits the input image into uniform tiles for computations to make simultaneous processing easier. Image classification as well as Object Recognition is performed as an experimentation on the Tiled Convolution technique. The Tile dimension, pattern, border extension is considered for accuracy on various Convolutional Neural Networks. Considering Network Image Classification, there are four networks considered namely, MobileNet, VGC16, Residual network-50 and Residual Network-18. For these networks, when the tile length is greater than 1 the border extension is asymmetrical. The CNN layers are altered such that the tile length is set to t succeeded by a pooling layer t by t. These network models are trained while the hyperparameter used to control the process of learning is kept constant for all these networks. Out of these networks, it is observed that three networks have an enhanced accuracy post modification. The study involves models are initially trained as well as models that tuned using prior trained models. It is observed that the Networks VGC16, Residual network-50 and Residual Network-18 obtained a higher and improved accuracy through tunning with prior trained models whereas MobileNet requires initial training from scratch. This method aims at learning optimal configurations of the Programmable Logic Devices for CNN. Machine learning techniques are algorithms are used for training model of CNN and corresponding Programmable Logic Devices. Resource location is optimized using Look up Tables. These tables play an essential role in configuration of Programmable Logic Devices, where the input in these tables represents binary values while the output consists of precomputed values for every combination of input values. This mechanism has three main phases before implementation which involves the Training Stage, Redundancy Reduction stage and the final phase of Logical enhancement. In the training stage, ‘Learning Programmable logic device configuration of neural networks’ uses machine learning methods for training the CNN model used in this study. Optimal configurations of Programmable logic devices are predicted for the CNN model that shows best accuracy in the Tiled Convolution. Every layer of the model has features β that are involved in the learning process combined with weights as well as sparsity is introduced for reduction of non-zero weights using $L2$ Regularizer, for which the training loss Υ is calculated using the equation given below:

$$\Upsilon = \kappa\left(\sum_{m=1}^{M}\sum_{d=1}^{D}\left(\varpi^{(m,d)}\right)^2\right)^{\frac{-1}{2}} \qquad (3)$$

In the equation (3), the sparsification factor is denoted as κ, count of layer given as M and the number of channels in every layer is given as D. The weighted vector for layers m and d is expressed as ϖ. The next phase in the proposed work includes the Redundancy Reduction stage. After the training of the neural network less significant or redundant configurations within the Programmable Logic Device is removed. Basically, the Look up table configurations are eliminated, this enhances the complete accuracy and performance of the device implementation. This process involves application of a threshold value denoted as ϑ for every weight ϖ. A relational link is developed in regard to accuracy and area utilization.

$$\varpi \leftarrow \begin{cases} \varpi & if\ \|\varpi\|\ is\ greater\ than\ the\ thresho \\ 0 & otherwi \end{cases} \quad (4)$$

The Convolutional Neural Network is then binarized for the recovery of accuracy loss that is induced. This proposed study uses XNOR gate instead of a L −Programmable Logic Device, that has an initial input of $\tilde{y}_1^{(n,u)}$ to retain the original value and dimension of the input connection, that withholds the Redundancy Reduced Binary Neural Network Structure. While $L - Q\ is\ greater\ than\ 1$, the corresponding inputs $L - Q - 1$ of the same tile of convolution $\tilde{y}_1^{(n,u)}$ that proves the dimension of the tile remains the same. Their selection is limited, where each connected input image is linked to at least one lookup table. For $Q\ is\ greater\ than\ 0$, Q is the final input for look up table that is linked to a $Q - bit\ memory, \tilde{q}^{(n,u)}$. The look up tables are limited in terms of adding look up table connections, this is resolved by using larger value of L and lower value of Q, in this case there will be lesser signals for the look up table. For this, the value of $L - Q$ has to be decreased to avoid loss on input.

For a binary domain of $\{-1,1\}^O$ along with Binary Neural Networks, there are constraints for training of models for operation on real vectors $\mathbb{S}^{\tilde{O}}$ for back propagation. This is resolved by interpolating expansion of $\hat{h}_n$: $\mathbb{S}^{L-Q} \times \mathbb{S}^Q \rightarrow \mathbb{S}$ where $\hat{h}_n(\tilde{y}^{(n,u)}, \tilde{q}^{(n,u)}) = h_n(\tilde{y}^{(n,u)}, \tilde{q}^{(n,u)})$ and $h_n: \{-1,1\}^{L-Q} \times \{-1,1\}^Q \rightarrow \{-1,1\}$ for h_n. Consider h_n to be a constant value, for a Boolean input, then $\hat{h}_n$ also remains unchanged. The expansion is performed using Lagrange interpolation that is expressed in the equation below:

$$\hat{h}_n(\hat{\tilde{y}}^{(n,u)}, \hat{\tilde{q}}^{(n,u)}) = \sum_{e\ belongs\ to\ \{-1,1\}^L} \left(\hat{d}_e \prod_{l=1}^{L} \left(\begin{bmatrix} \hat{\tilde{y}}^{(n,u)} \\ \hat{\tilde{q}}^{(n,u)} \end{bmatrix} - e_l \right) \right) \quad (5)$$

Considering the weights used in the training phase in equation (3), these weights are included in the retraining of model. The aim is the achieve the given below equation after the relinked signals are eliminated after Redundancy Reduction.

$$\hat{h}_n(\hat{\tilde{y}}^{(n,u)}, \hat{\tilde{q}}^{(n,u)}) = \sum_{j\ belongs\ to\ \{1,\dots,L-Q\}} \hat{\tilde{y}}_j^{(n,u)}, \hat{\varpi}_j^{(n,u)} \quad (6)$$

After all $\hat{h}_n$ is initialized, the second and last stage of retraining is performed after which the training constraints are binarized. High- accuracy training followed by Redundancy Reduction proves rapid convergence and also the initial stage of Took up table learning that decreased chances of overfitting. The enhancement of latency as well as area efficiency is achieved in this proposed study by enabling adequate memory resource use as well as faster performance with higher efficiency of the Programmable Logic Devices for Convolutional Neural Networks. The Concatenation of separate tiles result in the final output.

4. Performance Evaluation.

This section of the research presents the evaluation of model evaluation, which includes the evaluation and comparison with the existing model to prove the model efficiency. Moreover, in order to evaluate the model proposed model utilizes three dataset namely MNIST [23], CIFAR-10 [24] and ModelNet40 [25] dataset.

1.3 Evaluation on MNIST Dataset

The table1 presents a comparison of different methodologies for image classification, each associated with specific platforms, operating frequencies, LUT (Look-Up Table) counts, accuracy rates, and power efficiencies. Among these methods, several noteworthy observations can be made. "Re-Bnet" achieves a commendable accuracy of 98.29% using 25600 LUTs at a frequency of 200 MHz on the Spartan XC platform. Similarly, "FP-BNN" demonstrates a high accuracy of 98.24% on the Stratix-V platform, although its corresponding LUT count is unspecified. "BNN-PYNQ" attains an accuracy of 98.4% with 26809 LUTs at a frequency of 300 MHz on the Ultra96 platform, accompanied by a power efficiency of 267342. "Finn-R" stands at an accuracy of 97.69% using 38205 LUTs and 300 MHz frequency on the Ultra96 platform. "Finn," on the other hand, achieves an accuracy of 98.4% with 82988 LUTs at 200 MHz on the ZC706 platform. The "Proposed" methodology boasts the highest accuracy of 99.2% with a relatively modest LUT count of 29156, operating at 300 MHz on the Ultra96 platform. Impressively, this methodology maintains a power efficiency of 882190.

Table 1 MNIST dataset comparison

Methodologies	Platform	Frequency	LUT	Accuracy	Power Efficiency
Re-Bnet [26]	Spartan XC	200	25600	98.29	-

FP-BNN [27]	Stratix-V	150	-	98.24	-
BNN-PYNQ [28]	Ultra 96	300	26809	98.4	267342
FINN-R [29]	Ultra 96	300	38205	97.69	-
FINN[30]	ZC706	200	82988	98.4	-
EffiConv Net	Ultra 96	300	29156	99.2	882190

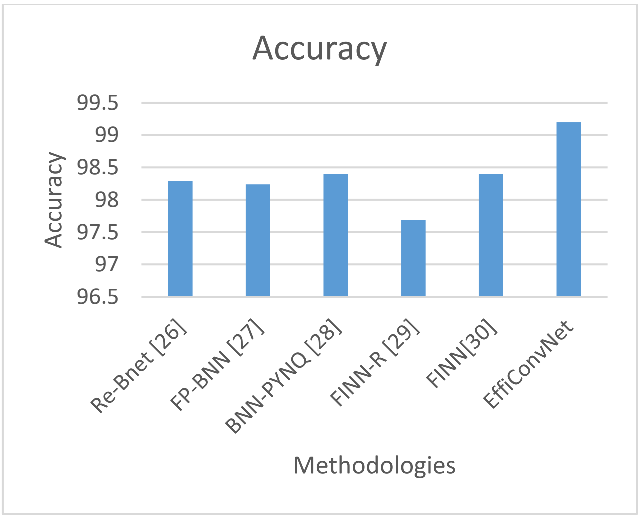

Fig 2 Accuracy comparison

The provided table compares several image processing methodologies based on key metrics like operating frequency, Look-Up Table (LUT) count, accuracy, processing speed (Kfps), and area efficiency."FINN-PYNQ" operates at 300 MHz with 25431 LUTs, achieving an 80.1% accuracy and processing speed of 1.9 Kfps. "ReBNet" achieves 80.59% accuracy at 200 MHz, utilizing 53200 LUTs and processing at 6 Kfps. "FBNA" has 88.61% accuracy with 26900 LUTs (frequency unspecified) and processes at 0.5 Kfps. "FINN-R" achieves 80.1% accuracy with 41733 LUTs at 300 MHz, processing at 19.5 Kfps. "Finn" obtains 80.1% accuracy at 125 MHz with 365963 LUTs, processing at 125 Kfps. The "Proposed" method reaches a 94.2% accuracy at 210 MHz using 290012 LUTs, processing at 205 Kfps.These findings highlight trade-offs between accuracy, processing speed, and LUT utilization. The "Proposed" approach strikes a balance, excelling in accuracy and processing speed while efficiently using resources. Method selection depends on specific application requirements.

1.4 Evaluation on CIFAR dataset

The table2 presents a concise analysis of various image processing methodologies based on key performance indicators. "FINN-PYNQ" achieves an 80.1% accuracy using 25431 LUTs at 300 MHz, processing at 1.9 Kfps with an area efficiency of 0.074. "ReBNet" operates at 200 MHz with 53200 LUTs, yielding 80.59% accuracy and processing at 6 Kfps, showing an area efficiency of 0.11. "FBNA" achieves an 88.61% accuracy with 26900 LUTs (frequency unspecified), operating at 0.5 Kfps and having an area efficiency of 0.02."FINN-R" and "FINN" both operate at 300 MHz, using 41733 and 365963 LUTs respectively, achieving 80.1% accuracy. "FINN-R" processes at 19.5 Kfps with an area efficiency of 0.467, while "FINN" achieves 125 Kfps with an area efficiency of 0.34 at 125 MHz. The "Proposed" method operates at 210 MHz with 290012 LUTs, attaining a high 94.2% accuracy and processing at an impressive 205 Kfps, accompanied by an area efficiency of 0.727.

Table 2 CIFAR dataset

Methodologies	Frequency	LUTs	Accuracy	Kfps	Area Efficiency
FINN-PYNQ [28]	300	25431	80.1	1.9	0.074
ReBNet [26]	200	53200	80.59	6	0.11
FBNA [31]	-	26900	88.61	0.5	0.02
FINN-R [29]	300	41733	80.1	19.5	0.467
FINN [30]	125	365963	80.1	125	0.340
EFFICONVNET	210	290012	94.2	205	0.727

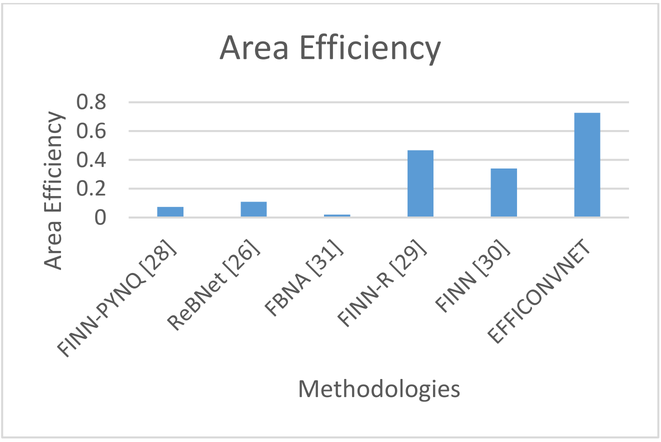

Fig 3 area efficiency comparison

1.5 Latency based Evaluation

The provided table offers a concise yet valuable comparison of different methodologies based on their accuracy and latency performance. "Pointnet++ [33]" achieves a commendable accuracy of 91.9%, albeit with a relatively higher latency of 117.59 units. On the other hand, "O-pointnet [34]" achieves a slightly lower accuracy of 88.5%, with the corresponding latency not specified, implying a potential trade-off between accuracy and latency. Notably, "EFFICONVNET" stands out by achieving an impressive accuracy of 93.9% while maintaining a significantly lower latency of 19.67 units.

Table 3 Accuracy and latency

Methodologies	Accuracy	Latency
Pointnet++[32]	91.9	117.59
O-pointnet [33]	88.5	
EFFICONVNET	93.9	19.67

O-pointnet [33] is deployed on a ZynqXC -7z045 device, although specific resource utilization details are not provided. Pointnet-FPGA [35] is implemented on a ZCU104 device and demonstrates variations in width and resource utilization. With a width of 16, it uses 30933 LUTs, 60412 Flip-Flops (FFs), 123 Block RAM (BRAM), and 1026 Digital Signal Processing (DSP) units. This methodology achieves a processing rate of 130 GOPS. When the width is reduced to 8, resource utilization is adjusted, with 19530 LUTs, 36010 FFs, 114 BRAM, and 1026 DSPs, enabling an increased processing rate of 182.1 GOPS. Similarly, point cloud [36] is executed on a ZCU104 device with a width of 16-8 (specific width not specified), utilizing 17614 LUTs, 12142 FFs, 365.5 BRAM, and 256 DSPs.On the other hand, EFFICONVNET stands out in terms of resource utilization, being implemented on a KCU150 device with a width of 8. It employs 57366 LUTs, 54082 FFs, 84.5 BRAM, and 2400 DSPs. Impressively, it achieves a significantly higher processing rate of 277.9 GOPS.

Methodologies	O-pointnet [33]	Pointnet-FPGA [34]	Pointnet-FPGA [34]	point cloud [35]	EFFICONVNET
Device	ZynqXC -7z045	ZCU 104	ZCU 104	ZCU 104	KCU150
width	16	16	8	16-8	8
LUT	-	30933	19530	17614	57366
FF	-	60412	36010	12142	54082
BRAM	-	123	114	365.5	84.5
DSP	-	1026	1026	256	2400
GOPS	1.28	130	182.1	17.73	277.9

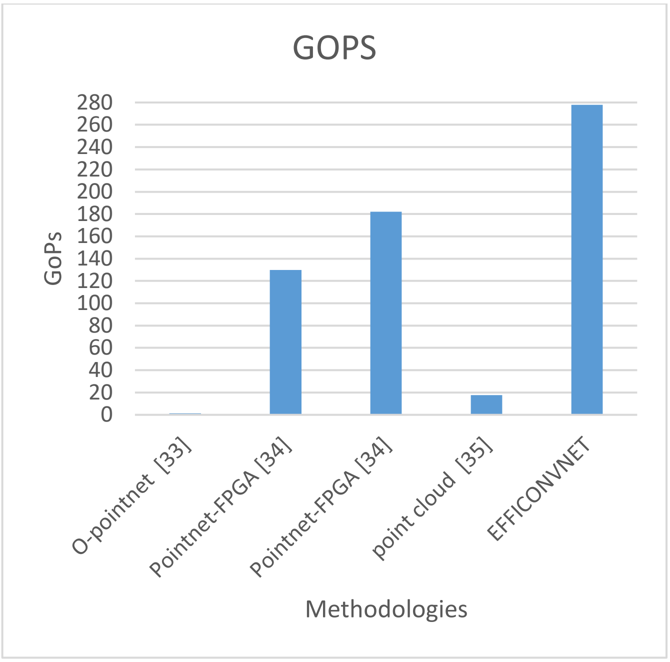

Fig 4 GOPS comparison

5. Conclusion

In the realm of deep learning acceleration on FPGA platforms, the pursuit of area efficiency and low latency has emerged as a crucial but often disjointed endeavor. EffiConvNet stands as an innovative paradigm that marries novel concepts such as Block Convolution, logic expansion, and ternary neural networks. Through this integration, the model offers an effective solution to the intricate challenges of efficiency in deep learning systems. The methodologies were evaluated using the MNIST, CIFAR-10, and ModelNet40 datasets.For the MNIST dataset, the "EFFICONVNET" methodology demonstrated the highest accuracy of 99.2% while maintaining an area efficiency of 882190. This approach struck a balance between accuracy and resource utilization on the Ultra96 platform. Similarly, on the CIFAR dataset, the "EFFICONVNET" approach achieved a remarkable 94.2% accuracy with an area efficiency of 0.727 at 210 MHz, highlighting its superior performance in both accuracy and efficiency. Furthermore, the latency-based evaluation showcased that "EFFICONVNET" achieved an impressive accuracy of 93.9% while maintaining a significantly lower latency of 19.67 units, indicating its effectiveness in balancing accuracy and responsiveness.

References

[1] P. Dhilleswararao, S. Boppu, M. S. Manikandan and L. R. Cenkeramaddi, "Efficient Hardware Architectures for Accelerating Deep Neural Networks: Survey," in IEEE Access, vol. 10, pp. 131788-131828, 2022, doi: 10.1109/ACCESS.2022.3229767.

[2] R. Sayed, H. Azmi, H. Shawkey, A. H. Khalil and M. Refky, "A Systematic Literature Review on Binary Neural Networks," in IEEE Access, vol. 11, pp. 27546-27578, 2023, doi: 10.1109/ACCESS.2023.3258360

[3] K. S. Zaman, M. B. I. Reaz, S. H. Md Ali, A. A. A. Bakar and M. E. H. Chowdhury, "Custom Hardware Architectures for Deep Learning on Portable Devices: A Review," in IEEE Transactions on Neural Networks and Learning Systems, vol. 33, no. 11, pp. 6068-6088, Nov. 2022, doi: 10.1109/TNNLS.2021.3082304

[4] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang, ‘‘Accelerating binarized convolutional neural networks with software-programmable FPGAs,’’ in Proc. ACM/SIGDA Int. Symp. FieldProgram. Gate Arrays, Feb. 2017, pp. 15–24

[5] K. Javeed, A. El-Moursy and D. Gregg, "EC-Crypto: Highly Efficient Area-Delay Optimized Elliptic Curve Cryptography Processor," in IEEE Access, vol. 11, pp. 56649-56662, 2023, doi: 10.1109/ACCESS.2023.3282781

[6] Y. Kuang et al., "ESSA: Design of a Programmable Efficient Sparse Spiking Neural Network Accelerator," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 30, no. 11, pp. 1631-1641, Nov. 2022, doi: 10.1109/TVLSI.2022.3183126.

[7] L. Guerra, B. Zhuang, I. Reid, and T. Drummond, ‘‘Automatic pruning for quantized neural networks,’’ 2020, arXiv:2002.00523.

[8] G. Anderson and C. P. Berg, ‘‘The high-dimensional geometry of binary neural networks,’’ 2017, arXiv:1705.07199.

[9] T. Fujii, S. Sato, and H. Nakahara, ‘‘A threshold neuron pruning for a binarized deep neural network on an FPGA,’’ IEICE Trans. Inf. Syst., vol. 101, no. 2, pp. 376–386, 2018.

[10] Y. Xu, X. Dong, Y. Li, and H. Su, ‘‘A Main/Subsidiary network framework for simplifying binary neural networks,’’ in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2019, pp. 7154–7162.

[11] X. Xiao and Z. Wang, ‘‘AutoPrune: Automatic network pruning by regularizing auxiliary parameters,’’ in Proc. Adv. Neural Inf. Process. Syst., vol. 32, Dec. 2019, pp. 1–15

[12] Y. Li and F. Ren, ‘‘BNN pruning: Pruning binary neural network guided by weight flipping frequency,’’ in Proc. 21st Int. Symp. Quality Electron. Design (ISQED), Mar. 2020, pp. 306–311.

[13] T. Geng, A. Li, T. Wang, C. Wu, Y. Li, R. Shi, W. Wu, and M. Herbordt, ‘‘O3BNN-R: An out-of-order architecture for high-performance and regularized BNN inference,’’ IEEE Trans. Parallel Distrib. Syst., vol. 32, no. 1, pp. 199–213, Jan. 2021.

[14] J. Gao, Q. Liu, and J. Lai, ‘‘An approach of binary neural network energyefficient implementation,’’ Electronics, vol. 10, no. 15, p. 1830, Jul. 2021

[15] Q. Wu, X. Lu, S. Xue, C. Wang, X. Wu, and J. Fan, ‘‘SBNN: Slimming binarized neural network,’’ Neurocomputing, vol. 401, pp. 113–122, Aug. 2020.

[16] B. -K. Yan and S. -J. Ruan, "Area Efficient Compression for Floating-Point Feature Maps in Convolutional Neural Network Accelerators," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 70, no. 2, pp. 746-750, Feb. 2023, doi: 10.1109/TCSII.2022.3213847.

[17] S. Wiedemann et al., "FantastIC4: A Hardware-Software Co-Design Approach for Efficiently Running 4Bit-Compact Multilayer Perceptrons," in IEEE Open Journal of Circuits and Systems, vol. 2, pp. 407-419, 2021, doi: 10.1109/OJCAS.2021.3083332.

[18] J. Wu et al., "An Energy-Efficient Deep Belief Network Processor Based on Heterogeneous Multi-Core Architecture With Transposable Memory and On-Chip Learning," in IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 11, no. 4, pp. 725-738, Dec. 2021, doi: 10.1109/JETCAS.2021.3114396.

[19] S. Liu, H. Fan, M. Ferianc, X. Niu, H. Shi and W. Luk, "Toward Full-Stack Acceleration of Deep Convolutional Neural Networks on FPGAs," in IEEE Transactions on Neural Networks and Learning Systems, vol. 33, no. 8, pp. 3974-3987, Aug. 2022, doi: 10.1109/TNNLS.2021.3055240.

[20] P. Plagwitz, F. Hannig, M. Ströbel, C. Strohmeyer, and J. Teich, ‘‘A safari through FPGA-based neural network compilation and design automation flows,’’ in Proc. IEEE 29th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM), May 2021, pp. 10–19.

[21] M. Blott, N. J. Fraser, G. Gambardella, L. Halder, J. Kath, Z. Neveu, Y. Umuroglu, A. Vasilciuc, M. Leeser, and L. Doyle, ‘‘Evaluation of optimized CNNs on heterogeneous accelerators using a novel benchmarking approach,’’ IEEE Trans. Comput., vol. 70, no. 10, pp. 1654–1669, Oct. 2021

[22] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, ‘‘Quantization and training of neural networks for efficient integer-arithmetic-only inference,’’ in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Jun. 2018, pp. 2704–2713. Y. Lecun. MNIST Datset. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[23] A. Krizhevsky. CIFAR-10 Dataset. [Online]. Available: https://www.cs. toronto.edu/~kriz/cifar.html

[24] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1912–1920, 2015

[25] M. Ghasemzadeh, M. Samragh, and F. Koushanfar, ``ReBNet: Residual binarized neural network," in Proc. IEEE 26th Annu. Int. Symp. Field- Program. Custom Comput. Mach. (FCCM), Apr. 2018, pp. 57_64.

[26] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, ``FP-BNN: Binarized neural network on FPGA," Neurocomputing, vol. 275, pp. 10721086, Jan. 2018.

[27] Xilinx Research Labs. (2017). BNN-PYNQ. [Online]. Available: https://github.com/Xilinx/BNN-PYNQ.

[28] M. Blott, T. B. Preuÿer, N. J. Fraser, G. Gambardella, K. O'brien, Y. Umuroglu, M. Leeser, and K. Vissers, ``FINN-R: An end-to-end deeplearning framework for fast exploration of quantized neural networks," ACM Trans. Recongurable Technol. Syst., vol. 11, no. 3, pp. 1-23,Sep. 2018.

[29] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, ``FINN: A framework for fast, scalable binarized neural network inference," in Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays, Feb. 2017, pp. 6574.

[30] M. Ghasemzadeh, M. Samragh, and F. Koushanfar, ``ReBNet: Residual binarized neural network," in Proc. IEEE 26th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM), Apr. 2018, pp. 5764.

[31] P. Guo, H. Ma, R. Chen, P. Li, S. Xie, and D.Wang, ``FBNA: A fully binarized neural network accelerator," in Proc. 28th Int. Conf. Field Program. Log. Appl. (FPL), Aug. 2018, pp. 511513.

[32] J. Kim, J. K. Lee, and K. M. Lee, "Accurate image super-resolution using very deep convolutional networks," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 1646–1654.

[33] N. Suda et al., "Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks," in Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays, 2016, pp. 16–25.

[34] Y. S. Shao et al., "SIMBA: Scaling deep-learning inference with multichip- module-based architecture," in Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchit., 2019, pp. 14–27.

[35] Y. Ma, Y. Cao, S. Vrudhula, and J.-S. Seo, "Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks," in Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays, 2017, pp. 45–54.

[36] V. Maruthi Prasad, B. Bharathi, "A Novel Trust Negotiation Protocol for Analysing and Approving IoT Edge Computing Devices Using Machine Learning Algorithm", International Journal of Computer Networks and Applications (IJCNA), 9(6), PP: 712-723, 2022, DOI: 10.22247/ijcna/2022/217704.

[37] Prasad, V. M. ., & Bharathi, B. (2023). Security in 5G Networks: A Systematic Analysis of High-Speed Data Connections. International Journal on Recent and Innovation Trends in Computing and Communication,11(5),216–222. https://doi.org/10.17762/ijritcc.v11i5.6608