IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Review Of CNN Architectures And Challenges In AI Application

Prakash Kumar
Sr. Manager Physical design Tech Mahindra
Bangalore, India

Anshuj Jain
Electronics and communication
RNTU
Bhopal, India

***Abstract*—This paper explores the progressive evolution of Convolutional Neural Network (CNN) architectures for Field-Programmable Gate Arrays (FPGAs), emphasizing design optimization strategies, advancements in computational complexity reduction, speed enhancements, and efficient resource utilization. Recent developments in FPGA-based CNN architectures are reviewed, highlighting key innovations and methodologies that have contributed to achieving real-time inference on resource-constrained hardware.**

***Keywords*— CNN, FPGA, CPU, GPU, RAM**

I. INTRODUCTION

Neural networks are a type of machine learning algorithm that are inspired by the human brain. They are made up of interconnected nodes, or neurons, that learn to recognize patterns in data. Neural networks have been used to achieve state-of-the-art results in a wide range of tasks, including image recognition, natural language processing, and speech recognition. One of the key advantages of neural networks is their ability to learn from data. They can be trained on large amounts of data to recognize complex patterns that would be difficult to identify using traditional methods. This makes them well-suited for tasks where there is a lot of data available, such as image recognition and natural language processing [1].

Neural networks are also very powerful. They can be used to solve problems that are difficult or impossible to solve with traditional methods. For example, neural networks have been used to develop self-driving cars and to diagnose diseases. However, neural networks also have some disadvantages. They can be difficult to train, and they can be sensitive to noise in the data. Additionally, they can be computationally expensive to run [2].

Convolutional Neural Networks (CNNs) have brought about remarkable breakthroughs in various domains, such as image recognition, natural language processing, and medical diagnosis. These networks have demonstrated state-of-the-art performance by automatically learning hierarchical features from raw data. However, the computational complexity of CNNs poses a significant challenge, especially when it comes to real-time inference on resource-constrained devices such as mobile phones, IoT devices, and embedded systems. To address this challenge, researchers and engineers are increasingly focusing on developing specialized hardware accelerators to efficiently execute CNN computations.

Traditionally, CNN inference has been performed on general-purpose computing platforms such as CPUs and GPUs. While these platforms offer significant processing power, they are not always well-suited for real-time and power-efficient inference tasks. As the demand for deploying CNNs in applications with strict latency and energy requirements grows, there is a need for dedicated hardware accelerators that can perform CNN computations with optimal performance and energy efficiency.

The motivation behind this paper lies in the pressing need to bridge the gap between the computational demands of CNNs and the available hardware resources. There are several key motivations for exploring recent advances in CNN architectures for hardware accelerator implementation:

a. Real-time Inference: Many applications, such as autonomous vehicles, robotics, and augmented reality, require real-time inference to make timely decisions. Hardware accelerators can significantly reduce the inference time, enabling these applications to operate smoothly and efficiently.
b. Energy Efficiency: CNN inference on power-constrained devices, including mobile phones and edge devices, often leads to high energy consumption. Hardware accelerators designed with energy efficiency in mind can prolong the battery life of these devices and reduce their environmental impact.
c. Scalability: As the diversity and complexity of CNN applications increase, so does the need for scalable solutions. Hardware accelerators can be customized to meet the specific requirements of different applications, ensuring optimal performance across a wide range of use cases.
d. Compact and Lightweight Models: CNN models are becoming increasingly complex, which poses challenges in terms of memory and storage requirements. Specialized hardware can handle

model complexity by offloading computations and enabling the deployment of compact, lightweight models.

e. Edge Computing: The rise of edge computing has brought attention to the need for on-device processing, reducing the reliance on cloud resources. Hardware accelerators enable efficient on-device inference, preserving privacy, reducing latency, and minimizing data transfer.

II. EVOLUTION OF CNN ARCHITECTURES FOR FPGAS

For different applications of image classification CNN is a widely applied machine learning algorithm/architecture. The block diagram of CNN algorithm implementation on FPGA is shown in Fig. 1.

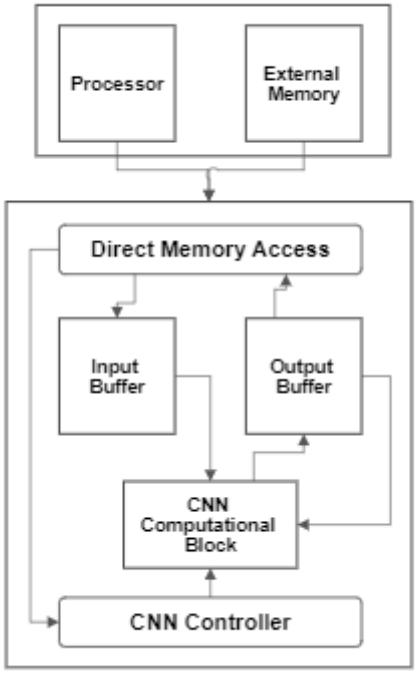

Fig 1. FPGA Implementation of CNN Architecture

There are many CNN-based architectures, which have been classified as early FPGA-based architectures and Compact CNN architectures discussed in the section, highlighting their characteristics.

A. Early FPGA-based CNN Architectures

SqueezeNet: It was presented by Iandola et al. in 2016, and focuses on reducing model size and computational complexity without sacrificing accuracy [1]. It introduces 1x1 convolutions (squeeze layers) to efficiently reduce the number of input channels, followed by a combination of 3x3 and 1x1 convolutions (expand layers) to capture rich feature representations. SqueezeNet's "squeeze and expand" architecture balances computational efficiency and accuracy. The 1x1 convolutions significantly reduce the number of parameters, enabling the network to be compressed while maintaining performance. This makes SqueezeNet an attractive choice for embedded systems, with limited memory and computation resources.

MobileNet: Introduced by Howard et al. in 2017, is designed to provide efficient convolutional operations by leveraging depthwise separable convolutions [2]. Traditional convolutions are split into two separate layers: depthwise convolution, which applies a single convolutional filter per input channel, and pointwise convolution, which performs 1x1 convolutions to combine information across channels. This separation significantly reduces computational complexity while retaining expressive power. MobileNet's depthwise separable convolutions allow for efficient computation by first capturing spatial information and then combining it across channels. This design reduces the number of multiplications, making it well-suited for embedded systems. The architecture achieves a good trade-off between accuracy and speed, making it suitable for real-time applications on FPGAs.

ShuffleNet: It was proposed by Zhang et al. in 2018, and introduces channel shuffling and group convolutions to minimize computation while maintaining information flow across channels [3]. The architecture divides the input channels into groups, applying separate convolutions to each group, and then shuffling the output channels to mix information. By using group convolutions and channel shuffling, ShuffleNet reduces the computation cost associated with cross-channel interactions. This design enables efficient feature extraction while achieving competitive accuracy. The architecture's adaptability to varying hardware constraints makes it a strong candidate for embedded FPGA implementations.

EfficientNet: It was proposed by Tan et al. in 2019, and introduces a compound scaling method that optimizes model depth, width, and resolution simultaneously [4]. The architecture uses a baseline network and scales it uniformly to achieve better performance across different resource constraints. EfficientNet addresses the challenge of balancing depth, width, and resolution by using a compound scaling strategy. By optimizing these dimensions together, the architecture achieves higher accuracy with fewer parameters compared to handcrafted designs. This adaptability to different computational budgets makes EfficientNet a valuable candidate for FPGA-based embedded systems.

B. Compact CNN Architectures

Xception (2016) (Extreme Inception) is an architecture introduced by Chollet in 2016 [5]. It extends the Inception architecture by replacing standard convolutions with depthwise separable convolutions across all layers. This design leads to increased efficiency and reduced computational complexity.

CondenseNet (2018) introduced by Huang et al. in 2018, presented a novel approach to network compression [6]. It utilizes a "learning from experts" paradigm, where a compact model learns from a larger, more expressive "teacher" network. The architecture dynamically selects important filters to keep during training, resulting in a compact yet efficient model.

MobileNetV2 (2018) a follow-up to MobileNet, was introduced in 2018 by Sandler et al [7]. It refines the depthwise separable convolution approach and introduces inverted residuals and linear bottlenecks. These enhancements further improve the efficiency and performance of the architecture.

PeleeNet (2018) proposed by Wang et al. in 2018, aims to strike a balance between accuracy and efficiency [8]. It introduces a novel Pelee block that combines multiple feature maps at different scales. This block enables the network to maintain accuracy while minimizing computation.

FBNet (2019) introduced by Wu et al. in 2019, utilizes a differentiable architecture search approach to discover efficient network architectures [9]. It introduces a flexible block-wise search space, enabling the network to be optimized for different hardware constraints.

MixNet (2020) introduced by Tan et al. in 2020, utilizes mixed-depthwise convolutions to improve efficiency [10]. It explores various combinations of depthwise and pointwise convolutions to optimize the network's performance and computational complexity. MixNet achieves a good balance between accuracy and efficiency.

High-Resolution Network (HRNet) (2020) proposed by Wang et al. in 2020, challenges the trade-off between spatial resolution and computation efficiency [11]. HRNet maintains high-resolution representations throughout the network's processing, allowing it to capture fine-grained details while still achieving competitive performance.

GhostNet (2020) proposed by Han et al. in 2020, introduces the concept of "ghost" modules, where a lightweight operation is performed on a subset of the input channels [12]. This approach reduces computational complexity while maintaining expressive power. GhostNet demonstrates strong performance on both accuracy and efficiency metrics.

RegNet (2020) proposed by Radosavovic et al. in 2020, introduces a design space exploration approach to find optimal architectures under computational constraints [13]. It focuses on scaling up networks while maintaining efficiency. By systematically searching the architecture space, RegNet identifies high-performing designs that match specific resource budgets.

C. Hardware Implementation Challenges

Despite their advantages, FPGA implementations of neural networks come with their challenges. Design complexity is a significant hurdle, as creating efficient FPGA architectures for diverse neural network models requires specialised knowledge and expertise. FPGAs operate under resource limitations, necessitating careful resource allocation to fully utilize available hardware. Managing the memory hierarchy, including on-chip memory, cache coherence, and memory bandwidth, presents a complex challenge in maintaining efficient data access. Balancing the need to reduce precision for efficiency while preserving model accuracy is challenging due to quantization-induced errors. Ensuring efficient data movement between memory hierarchies and processing units is critical to avoid performance bottlenecks. Adapting neural network algorithms to align with FPGA hardware and dealing with limited development tools and debugging support can slow down the development process. Integrating FPGA-accelerated neural networks with existing systems and software also poses integration complexities. Ensuring the correctness, reliability, and scalability of FPGA-based neural network implementations adds to the development effort. Lastly, the dynamic nature of workloads and neural network models requires flexibility in FPGA design and adaptation to changing requirements. While FPGAs hold immense promise for neural network acceleration, addressing these challenges is crucial for realizing their full potential.

III. Evaluation Metrics and Optimization Strategies

A. Performance Evaluation

1. Latency (s): Latency can be calculated by dividing the total processing time (in seconds) by the number of processed data points:

$$Latency\ (s) = \frac{Total\ Processing\ Time\ (s)}{Number\ of\ Data\ Points\ Processed} \quad — (1)$$

2. Throughput (OPS or IPS): Throughput is calculated as the reciprocal of latency:

$$\text{Throughput (OPS/IPS)} = \frac{1}{Latency\ (s)} \quad — (2)$$

3. Accuracy (%): Accuracy is typically measured as the ratio of correctly classified data points to the total number of data points:

$$Acuraccy = \frac{Number\ of\ Correctly\ Classified\ Data\ Points}{Total\ Number\ of\ Data\ Points} \quad — (3)$$

Training the model with quantization in mind can mitigate the impact of reduced precision on accuracy. Weight pruning techniques can reduce model complexity without significantly affecting accuracy.

4. Resource Utilization (%): Resource utilization is often reported as the percentage of FPGA resources used. This can vary depending on the specific resources of interest, such as logic elements, memory blocks, or DSP units:

$$Resource\ Utilization\ (\%) = \frac{Used\ FPGA\ Resources}{Total\ FPGA\ Resources}\ X\ 100\% \quad — (4)$$

Mapping layers to FPGA resources efficiently and optimizing resource-sharing can reduce resource usage. Design custom hardware modules tailored to specific CNN layers to optimize resource utilization.

5. Energy Efficiency (OPS/Watt): Energy efficiency measures the number of operations performed per watt of power consumed. It can be calculated as:

$$Energy\ Efficiency = \frac{Throughput}{Power\ Consumption} \quad — (5)$$

6. Memory Efficiency (%): Memory efficiency is often measured as the percentage of on-chip memory used compared to the total available on-chip memory:

$$Memory\ Efficiency = \frac{Used\ On-Chip\ Memory}{Total\ On-chip\ Memory}\ X\ 100\% \quad — (6)$$

Effective use of on-chip memory resources, such as block RAM, can reduce off-chip memory access, minimizing memory-related bottlenecks.

B. Hardware Optimization Strategies

a. Quantization and Pruning Techniques

Reducing the precision of weights and activations (quantization) or removing less important connections (pruning) can significantly reduce memory requirements and computational complexity. The benefits of these techniques are lower memory usage, reduced computation, and potentially faster inference. Challenges of these techniques are balancing between precision reduction and model accuracy, and handling quantization-induced degradation.

b. Layer Fusion

Combining consecutive layers with compatible operations into a single operation reduces memory access and intermediate data transfer. The benefits of these techniques are minimized memory bandwidth requirements, reduced latency, and efficient resource utilization. Challenges of these techniques are Identifying compatible layers, and managing fused operations for different layer sizes.

c. Winograd Transformation

Applying the Winograd convolutional transformation to reduce the number of multiplications required for convolution operations. The benefits of these techniques are faster convolutions, reduced arithmetic complexity, and improved efficiency. Challenges of these techniques are adapting the transformation to different convolutional kernel sizes, and handling the overhead of transformation computation.

d. Dataflow Optimization

Reordering computations to exploit parallelism and minimize data movement, enhancing memory access patterns. The benefits of these techniques are improved memory bandwidth utilization, reduced data transfer time, and increased throughput. The challenge of these techniques is

designing an optimal data flow for different CNN layers and architectures.

e. Loop Unrolling

Expanding loops in the code to expose more parallelism and minimize loop overhead. The benefits of these techniques are Increased instruction-level parallelism, reduced loop overhead, and improved throughput. Challenges of these techniques are Balancing between loop unrolling and resource usage, and managing code complexity.

f. Custom Hardware Units

Description: Designing custom hardware accelerators for specific operations (e.g., convolution, pooling) to maximize efficiency. The benefits of these techniques are Higher performance, lower power consumption, and optimized resource usage. Challenges of these techniques are Hardware design complexity and integration with existing FPGA resources.

g. Memory Hierarchy Optimization:

Employing on-chip memory (registers, block RAM) efficiently to reduce data movement to external memory. The benefits of these techniques are faster data access, reduced memory bandwidth usage, and improved latency. The challenges of these techniques are managing memory space, handling data storage and transfers, and minimizing memory conflicts.

h. Pipeline and Parallelism:

Partitioning the computation into stages and exploiting pipelining and parallelism to maximize resource utilization. The benefits of these techniques are higher throughput, reduced latency, and improved overall performance. Challenges of these techniques are ensuring correct data dependencies, managing pipeline stalls, and balancing resource allocation.

i. Layer Skipping and Skip Connections:

Introducing skip connections to allow skipping certain layers during inference based on certain conditions or heuristics. The benefits of these techniques are faster inference for specific inputs, reduced computational load, and potential energy savings. The challenges of these techniques are identifying suitable layers for skipping and avoiding adverse effects on accuracy.

IV. COMPARATIVE ANALYSIS

The reviewed architectures have been divided into two categories; CNN architecture on software and CNN architecture on FPGA. In Table 1 both categories have been compared with respect to Top-1 accuracy and number of trainable parameters. Top-1 accuracy gives the highest probability of classifying class A and class A. The number of parameters are the trainable parameters from each layer and it provides the number of learnable elements for a filter.

Table 1. Comparison of state-of-the-art models

Type of Model	Model Name	Top-1 Accuracy	No. of Parameters
Convolution Based Architecture on Software	DenseNet-121	75	8.1M
	Xception	79	22.9M
	EicientNet-B5	82.6	30.6M
	NasNetLarge	83.5	88.9M
	ConvNextLarge	86.3	197M
CNN Architecture for FPGA	SqueezeNet	60.4	1.2M
	MobileNet-V2	71	3.4M
	ShuleNet	72.5	3.4M
	GhostNet	73.9	5.2M
	EicientNet-B0	77.1	5.3M

It is evident from the comparison that Top-1 accuracy is proportional to the number of parameters, the larger the learnable parameters, the better the Top-1 accuracy. It has also been observed that the parameters are less for FPGA which may be due to the hardware resource constraints. Hence, the Top-1 accuracy has been compromised due to reduced parameters.

Similarly, Table.2 compares all the CNN-based architectures that have been implemented on FPGA. The common metrics such as accuracy, sensitivity, and F1-score have been compared along with the specific metrics such as Kappa score and MAC units.

Table 2. Comparison of the models on FPGA

Model	**Acc**	**Se**	**F1**	**Kappa**	**Params**	**MACs**
EfficientNet	95.7	94.4	94.0	93.7	2.91	683
MobileNetV2	95.4	92.6	93.3	93.1	3.32	313
ESPNetV2	95.0	92.7	93.0	92.6	2.19	297
ShuffleNet	94.9	91.6	92.5	92.3	5.32	570
MixNet-S	95.4	92.8	93.4	93.2	2.60	250
MDNet-S	96.9	95.1	95.5	95.3	2.50	220

It has been observed that MDNet-S is showing the best result in terms of accuracy and MAC units. The results and observations pointed out that the architecture to be deployed on an FPGA should be optimized in terms of computing hardwares.

V. Availability of data and materials

Table 1 and Table 2 in the present paper.

VI. Competing interests

As if know there is dedicated CNN architecture for each specified application. My work is to find efficient architecture for all image processing CNN architecture as generic

VII. Funding

Not Applicable

VIII. Authors' contributions

The reviewed architectures have been divided into two categories; CNN architecture on software and CNN architecture on FPGA. In Table 1 both categories have been compared with respect to Top-1 accuracy and number of trainable parameters. Comparison of the models on FPGA as shown in table 2.
And discussed their performance, accuracy Mac units and computing hardware. And understand there is still area of improvement to have generic solution for image processing for efficient performance

IX. Conclusion

In this work, different CNN based architectures have been reviewed which are compatible with hardware accelerators such as FPGAs. It has been observed that the development of new architectures may give better performance but not necessarily be an efficient architecture when deployed on the FPGA. It has been understood that machine learning architecture (such as CNN) can be implemented in two ways: a. RTL Implementation b. HW/SW Co-simulation. RTL implementation can be a better approach provided the architecture is simple or less complex. For any complex architecture (i.e. with more layers and backdrops) HW/SW co-simulation techniques (such as AutoML, NNgen etc.) are better. Study has also shown the development of compressed CNN architectures which addresses the hardware constraints of FPGA and propose an efficient implementation on FPGA. This paper has provided research possibilities and a roadmap towards development of FPGA friendly ML architectures.

References

[1] Iandola, Forrest N., et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size." arXiv preprint arXiv:1602.07360 (2016).

[2] Howard, Andrew G., et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." arXiv preprint arXiv:1704.04861 (2017).

[3] Zhang, Xiangyu, et al. "Shufflenet: An extremely efficient convolutional neural network for mobile devices." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.

[4] Tan, Mingxing, and Quoc Le. "Efficientnet: Rethinking model scaling for convolutional neural networks." International conference on machine learning. PMLR, 2019.

[5] Chollet, François. "Xception: Deep learning with depthwise separable convolutions." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

[6] Huang, Gao, et al. "Condensenet: An efficient densenet using learned group convolutions." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.

[7] Sandler, Mark, et al. "Mobilenetv2: Inverted residuals and linear bottlenecks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.

[8] Wang, Robert J., Xiang Li, and Charles X. Ling. "Pelee: A real-time object detection system on mobile devices." Advances in neural information processing systems 31 (2018).

[9] Wu, Bichen, et al. "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2019.

[10] Tan, Mingxing, and Quoc V. Le. "Mixconv: Mixed depthwise convolutional kernels." arXiv preprint arXiv:1907.09595 (2019).

[11] Wang, Jingdong, et al. "Deep high-resolution representation learning for visual recognition." IEEE transactions on pattern analysis and machine intelligence 43.10 (2020): 3349-3364.

[12] Han, Kai, et al. "Ghostnet: More features from cheap operations." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2020.

[13] Radosavovic, Ilija, et al. "Designing network design spaces." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2020.

[14] Zhang, Xiaoqing, et al. "Mixed-decomposed convolutional network: A lightweight yet efficient convolutional neural network for ocular disease recognition." CAAI Transactions on Intelligence Technology (2023).

[15] Nechi, Anouar, et al. "FPGA-based Deep Learning Inference Accelerators: Where Are We Standing?." ACM Transactions on Reconfigurable Technology and Systems (2023).