Accelerating Neural Networks on FPGAs: An Overview

Moiz Zaheer Malik
B.Eng. Electronic Engineering
Hochschule Hamm-Lippstadt
Lippstadt, Germany
moiz-zaheer.malik@stud.hshl.de

***Abstract*—Neural networks are now used in many applications such as image recognition, self driving cars, and language processing. However, they need a large amount of computing power and memory to work efficiently. Traditional processors like CPUs are not fast enough, and GPUs consume a lot of energy. FPGAs offer a good balance between speed, flexibility, and power efficiency. They can be reprogrammed to match the exact structure of a neural network and process data in parallel using custom hardware. This report gives an overview of how FPGAs accelerate neural networks, explains common techniques such as pipelining, parallelism, and quantization, and discusses key challenges like memory bandwidth and resource limitations. The goal is to show how FPGA based accelerators can make neural networks faster and more efficient for real time and edge applications.**

I. Introduction

As digital technologies continue to advance and the availability of reliable and trustworthy data increases, artificial intelligence (AI) and deep learning techniques are gaining widespread popularity [29]. These technologies have demonstrated remarkable effectiveness in solving complex problems that were once considered beyond computational reach [29].

However, deep learning models particularly convolutional neural networks (CNNs) require extremely high computational power and memory bandwidth [29]. Such demands are difficult for traditional central processing units (CPUs) to meet efficiently, often resulting in limited performance [29]. To overcome these challenges, hardware accelerators such as application specific integrated circuits (ASICs) and field-programmable gate arrays (FPGAs) have emerged as key solutions for improving the processing speed of AI applications, including CNNs [29].

Among these, FPGAs are widely used to accelerate deep learning tasks because they excel at executing multiple operations in parallel and offer significantly better power efficiency compared to general purpose processors (GPPs) [29].

To make results more accurate in real time like in self driving cars and robots CNNs have to get bigger by adding more layers [48]. This makes them more powerful but also much heavier to process, with billions of calculations and millions of parameters that need strong computing power for training and testing [42, 8, 29].

Because of such massive computational requirements it create massive challenges for traditional general purpose processors (GPPs) [29]. As a result, hardware accelerators such as ASICs, GPUs, and FPGAs are increasingly used to improve the performance and outputrate of CNNs [29].

But still, there are some drawbacks between these platforms. GPUs, for instance, are widely used for CNN acceleration in both training and inference due to their high memory bandwidth and efficient parallel computation capabilities [66, 28, 60, 29]. But yet, GPUs high power consumption makes them less suitable for cloud based systems or battery powered CNN applications [19, 29].

In contrast, FPGAs have emerged as a preferred platform because they offer a balance between performance and energy efficiency [29]. Experimental results show that FPGAs deliver higher power efficiency (performance per watt) [29], than other accelerators, despite having relatively limited I/O bandwidth and computing resources compared to GPUs. They can still achieve moderate performance with significantly lower power consumption [38, 29].

ASICs can achieve high throughput by customizing memory hierarchies and dedicating resources to specific tasks [14, 29] . However, they suffer from long development cycles, high costs, and low flexibility in deep learning applications [23, 13, 29]. As an alternative, FPGA based accelerators provide high throughput at a reasonable cost, with low power consumption and reconfigurability [58, 45].

Another major advantage of modern FPGAs is the availability of High Level Synthesis (HLS) tools that allow developers to program hardware using C or C++, significantly simplifying the design process and reducing development time for FPGA based accelerators [19, 29].

In this report, we first explain what an FPGA is and how its internal structure allows parallel data processing. Then we describe what a neural network is and how it works through training and inference. After that, we discuss why FPGAs are suitable for running neural networks compared to other hardware like CPUs, GPUs, and ASICs. The next sections show how FPGAs accelerate neural networks using techniques such as pipelining, parallelism, and quantization, followed by real case studies from recent research. Finally, the report talks about current challenges that limit FPGA performance and ends with a conclusion summarizing the main findings and future directions.

II. What is an FPGA?

Field Programmable Gate Arrays (FPGAs) are special programmable chips they are off-the-shelf programmable devices which means FPGAs can be bought ready made and one just has to configure them for their own purpose, They can customized to perform many different hardware functions [29]. Unlike normal processors that follow fixed instructions, an FPGA can be configured to perform like any digtal circuit depending how its programmed. this is what makes it very usefull and flexible for application that require high speed and custom design [62].

The figure 1 Shows the basic structure of an FPGA. Inside an FPGA there are thousands of small building blocks called configuranle Logic Blocks (CLBs) that contain look Up Tables (LUTs) and Flip Flops (FFs) [29]. These are connected through programmable interconnection network, allowing signals to move freely between them. Around the edges of the chip, there are Input/Output (I/O) cells that connect the FPGA to external devices [27, 29].

FPGAs also include Digital Signals Processing (DSP) Block for fast Math operations, Block RAM (BRAM) for temporary data storage, and clock managment unites to keep every thing synchronized. Some even include high speed communication interfaces, making them great for data intensive applications [59, 34, 29].

Because of such structure, FPGAs can handle a large amount of parallel processing, meaning they can run many small task at once insted of running them one by one [29]. This makes the ideal for deep learning and neural network accelertion, where thousands of operations happen every second [15]. They are also ver power efficient, giving high performance for much less energy compared to CPUs or GPU [34, 29].

Another very strong feature of FPGAs is partial reconfiguration, which allows part of the chip to be reprogrammed while the rest keeps running and this is somthing very helpfull for deep learing models where one layer can be reloaded while another is still processing [29].

Recently, Programming models like OpenCL and tools such as High Level Synthesis (HLS) have made FPGA development easier [29]. They let developers use familiar languages such as C and C++ to desing hardware, reducing desing time and cost [39, 53, 43, 64].

III. What is a Neural Network?

Deep learning is becoming very popular and one of the important tools for solving complex problem because of the avilabilty of big data and also because of its computational capaibilties [44]. the domains the deep learing is to solve problem include image recognition [33], speech processing [4, 18, 25] , natural language processing [10], language translation [12], and autonomousvehicles [35, 44].

The popular algorithmic approch for deep learning Convolutional neural networks is imerging as the best in most of the domains. CNN can be dedvided into two different parts

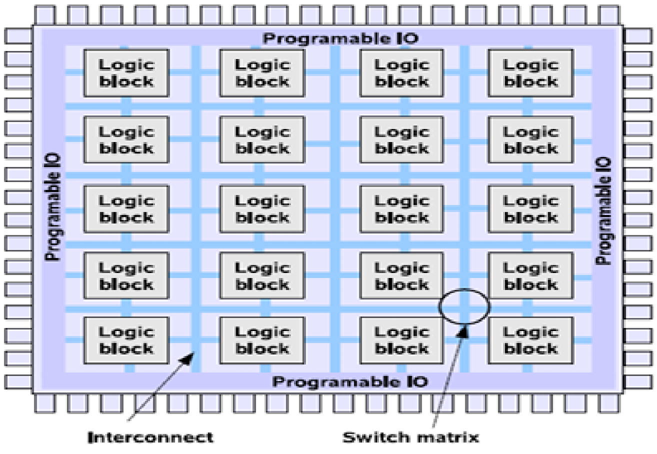

Fig. 1: FPGA Basic Structure [29, 62]

1) Training: During training, a neural network learns from a large number of datasets by adjusting its internal parameters, known as weights [44]. A deep learning engineer designs the network architecture deciding how many layers it has, what operations each layer performs, and how the layers are connected [44]. These weights control how strongly one neuron influences the next, determining what kind of features the layers detect, such as edges, colors, or shapes in images.

The goal of training is to find the best possible values for those weights. This is done using a mathematical optimization method called Stochastic Gradient Descent (SGD) [44].

The Training happens in three main steps. Forward propagation where the input goes through the network layer by layer to produce an output. next step is Error calculation where the network checks how much different the output is from the correct soulution and the difference here is called Error or Loss. Then in the Back Propagation the error is sent back through all the layers to update weights slightly to reduce the error next time

2) Inference: After the completion of training the, the networks is put into the real world. Where it takes new input data and makes predictions or classificatons[44]. This is the stage where a trained neural network is used to make predictions. Unlike training, it only performs forward propagation data passes through the layers to produce an output, without updating any weights. However, inference can still require massive computational power, especially for deep networks with hundreds of layers or for large inputs like high definition video [44]. Because many applications such as self driving cars and smart devices run on limited energy, achieving high energy efficiency during inference is crucial.

Neural networks use activation functions to introduce nonlinearity into the model [44]. One of the most common is the Rectified Linear Unit (ReLU), which sets all negative values to zero. The resulting values, called activations, represent the output of a layer and serve as the input to the next layer [44]. In practice, a large portion of these activations often between 50% and 70% become zero, which reduces the amount of

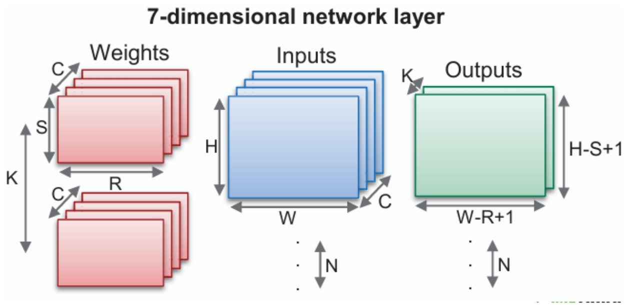

Fig. 2: Basic structure of a convolutional layer showing input activations, learned filter weights, and resulting output activations [44]

meaningful data passed forward and influences later hardware optimization strategies [44].

To illustrate the scale of modern neural networks, Table I summarizes three well known architectures. Each contains millions of parameters and requires billions of multiplication operations for a single inference, highlighting the need for efficient hardware implementations [44].

TABLE I: Characteristics of popular CNN architectures [44].

Network	Conv. Layers	Weights (MB)	Multiplies (B)
AlexNet	5	1.73	0.69
GoogLeNet	54	1.32	1.10
VGGNet	13	4.49	15.3

Convolutional Neural Networks (CNNs) are composed of multiple layers arranged in a sequence, forming a cascade of feature extraction and transformation steps [35]. These layers include convolutional layers that apply small filters (1×1, 3×3, or 5×5) that slide across the image and multiply with small parts and each filter detects specific patterns like vertical lines, texture or edges, non linear activation layers such as ReLU that introduce nonlinearity, and pooling layers that reduce the spatial resolution while preserving important information [44]. The weights that are learned during training, enabling the network to detect edges, shapes, and complex visual features [44]. In deeper networks, fully connected layers near the end combine all extracted features to perform classification. The output of one layer, known as an activation map, serves as the input for the next layer, allowing hierarchical feature learning from simple to complex representations [44].

As illustrated in Fig. 2, each convolutional layer in a neural network applies a small filter (set of weights) across the input feature maps to generate output activations. This operation extracts spatial features such as edges or textures and passes them to deeper layers for higher level representation [44].

IV. Why Use FPGA for Neural Networks?

One of the important subsets of artificial intelligence, Machine Learning (ML), focuses on algorithms that learn from large datasets to predict outcomes and perform tasks autonomously without explicit programming [44]. Recent research has achieved remarkable progress in domains such as image segmentation [52], object classification [33, 26] and detection [6], data classification [50], natural language processing (NLP) [16], edge computing [40], large scale scientific computing [20], and even for circuit design and optimization [51]. To achieve higher accuracy, ML models have become increasingly deep and complex, often containing redundant parameters that significantly increase computational and memory requirements [3, 24].

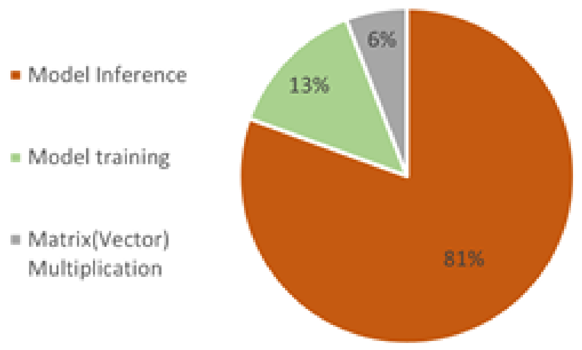

Fig. 3: Distribution of FPGA based ML research showing that inference dominates due to low latency and energy efficient characteristics of FPGAs [3]

As a result, ML inference and training demand massive processing power and memory bandwidth [3]. Traditional computing platforms such as Central Processing Units (CPUs) and Graphics Processing Units (GPUs) are widely used for ML but have notable limitations. CPUs are optimized for general purpose, mostly sequential tasks, making them inefficient for highly parallel ML workloads [3]. GPUs, while powerful for parallel operations, consume substantial energy and generate heat, posing challenges for portable or energy constrained devices [3].

Therefore, custom hardware architectures specifically designed for ML algorithms are becoming essential [3]. Field Programmable Gate Arrays (FPGAs) are particularly suited for this role because of their reconfigurable architecture. Their internal logic blocks, interconnections, and memory organization can be modified to match the structure of a neural network, even during runtime, allowing them to adapt dynamically to changing workloads [3]. This adaptability, combined with fine-grained parallelism and energy efficiency, makes FPGAs an ideal choice for both small scale edge computing and large scale cloud acceleration of ML tasks. Moreover, FPGAs are well suited for real time neural network inference because their hardware can be customized to the model structure, reducing latency and improving performance [3].

Another key advantage of FPGAs in neural network acceleration is their ability to deliver low latency inference [3]. Many real time applications, such as autonomous driving and video analytics, require response times within milliseconds [3].

FPGAs are good at this because they can run many things at the same time using parallel circuits and pipelined data paths that keep the work flowing smoothly [3, 41, 47, 49]. They also move data inside the chip very efficiently, which saves time by reducing how often they need to access slower external memory [3, 17, 67, 63]. This characteristic explains why a majority of FPGA based ML studies (approximately 81%) focus on inference rather than training, as illustrated in Fig. 3 [3].

Energy efficiency is another important reason to use FPGAs. They use less power because the hardware can be designed exactly for what the algorithm needs, so there is no wasted work or extra data movement [3, 9, 32, 55, 21]. Their streaming dataflow design and built in power control help reduce energy use even more [3]. They can also use smaller data types, like 8-bit numbers instead of 32-bit, and reuse memory inside the chip instead of reading it from outside. This makes them great for devices that run on batteries or for data centers where saving energy is important [3]. Compared to GPUs, FPGAs are more energy efficient, and unlike ASICs, they can still be reprogrammed for different models [57, 69, 3].

Neural networks also fit well with the structure of FPGAs. A lot of neural network work involves math like convolutions and matrix multiplications, and FPGAs can run many of these calculations in parallel using their DSP blocks. This makes them fast, efficient, and flexible for different types of ML models [3, 11].

Overall, FPGAs are powerful because they combine speed, flexibility, and low power use. They can handle real time tasks, save energy, and adapt to new models easily, making them a very good choice for accelerating neural networks.

V. How FPGAs Accelerate Neural Networks

FPGAs speed up neural networks by creating hardware that matches how these networks actually work. Instead of running all operations step by step like a CPU or using fixed parallel units like a GPU, an FPGA can be reconfigured to build custom pipelines for each layer. These pipelines perform many multiply-and-add operations at the same time, keep data close in on-chip memory, and reuse it to avoid slow transfers to external memory. Techniques such as parallel processing, pipelining, quantization, and data reuse all work together to increase performance and reduce power use [5, 1, 57, 61]. The following sections explain how these techniques are used in real FPGA designs.

A. Architecture Examples and Case Studies

A Special custom FPGA accelerator circuite was built for VGG 16 neural network which is used for image recognition it had 3,136 MAC units which are small blocks that do multiply and add operations and 14 input buffers these are used to store image pixels temporarily before processing it [29].

The design was smart because it reused pixels and weights insted of reading the same data again and again from memory, it shared them among Pof and Pix Piy MAC units

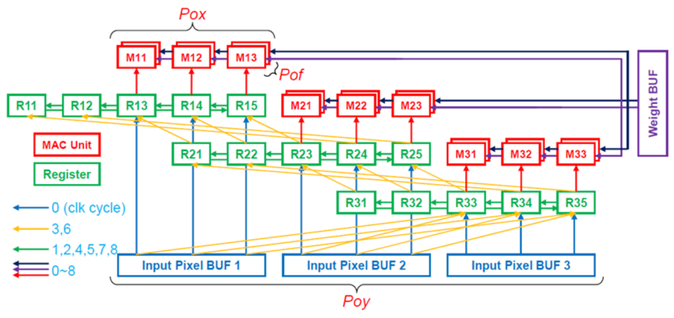

Fig. 4: VGG-16 FPGA Accelerator [29, 65]

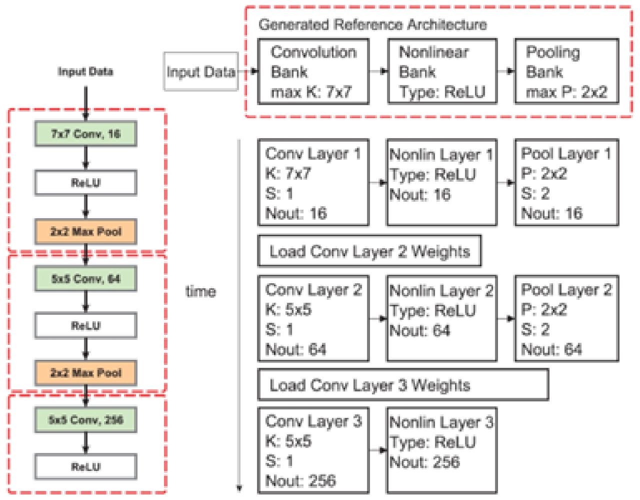
Input Data
7x7 Conv, 16
ReLU
2x2 Max Pool
5x5 Conv, 64
ReLU
2x2 Max Pool
5x5 Conv, 256
ReLU
Input Data
Generated Reference Architecture
Convolution Bank max K: 7x7
Nonlinear Bank Type: ReLU
Pooling Bank max P: 2x2
time
Conv Layer 1 K: 7x7 S: 1 Nout: 16
Nonlin Layer 1 Type: ReLU Nout: 16
Pool Layer 1 P: 2x2 S: 2 Nout: 16
Load Conv Layer 2 Weights
Conv Layer 2 K: 5x5 S: 1 Nout: 64
Nonlin Layer 2 Type: ReLU Nout: 64
Pool Layer 2 P: 2x2 S: 2 Nout: 64
Load Conv Layer 3 Weights
Conv Layer 3 K: 5x5 S: 1 Nout: 256
Nonlin Layer 3 Type: ReLU Nout: 256

Fig. 5: DLA with Double Buffering [29, 61]

as shown in the 4 [29]. This helps save time and memory bandwidth scince accessing memory is slow.

Because of that the system reaches 645.25 GPOS which means it could do 645 billion operations per second this is three times faster than earlier versions [29, 54, 46].

Aydonat et al [5]. presented a different kind of FPGA based accelerator that was based on OpenCL, which is high level programing language for hardware [29]. Their main goal was to use less external memory like DDR or SDRAM [29]. It was possible by caching feature maps directly on the FPGa, so they didn't need to fetch them from slow external memory, using stream buffers and double buffering while one layer is being processed the data for the next layer is alredy being loaded so there is o waiting time. [29]. Winograd algorithm was also used a clever math trick that that reduces the number of multiplications needed in convolution which helps in speeding things up even more 5 Depicts the Deep Learning Accelerator (DLA) based on OpenCL, including stream buffers, double buffered weight caches, and PEs [29].

Lu et al [36]. further improved version used a 2D Winograd pipeline with four stages (transform, multiply, inverse transform, accumulate), achieving 2.94 TOPS with only 23.6 W power showing that FPGAs can deliver both high speed and low energy use, as shown in 6 input lines are reused from on

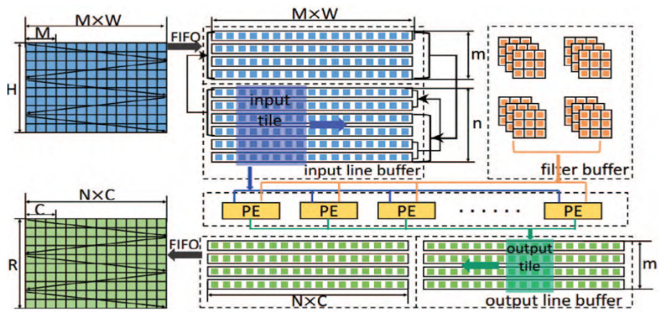

Fig. 6: Two dimensional Winograd accelerator showing circular buffer reuse and four stage pipelined computation transform, element wise multiplication, inverse transform, and accumulation [29, 36]

chip buffers while new data are fetched from off chip memory in parallel, overlapping transfer and computation [29].

B. Common FPGA Acceleration Techniques

1) Parallelism: Parallelism means doing many things at the same time insted of doing them one by one. In CNN, a convolution layer repeates the same operation many times for different pixel and filters. On an FPGA, many copies can be made of these operations so they work simultaneously [29]. and this process is known as loop unrolling. Each unrolled loop generates multiple MAC units that work simultaneously, increasing throughput. However, higher parallelism also consumes more FPGA resources, so designers must balance performance with hardware usage [29].

2) Pipelining: Pipelining keeps all parts of the circuit active by dividing computations into sequential stages. In the Winograd engine, data flow through four stages transform, multiply, inverse transform, and accumulate [29]. At the same time, double buffering allows new data to be loaded while the current data are still being processed. This overlap removes idle cycles and maintains high performance [29].

3) Quantization: Quantization reduces the number of bits used for computation. The FINN framework converts networks into binary versions that use only 1-bit weights and activations. Multiplications are replaced by simple XNOR and popcount operations. Batch normalization and activation are done using threshold comparisons, which saves both power and hardware resources [57].

4) Data Reuse and Local Memory: Accessing off chip DRAM takes a lot of energy and time. To avoid this, FPGA accelerators store feature maps and buffering are used to prefetch data so that computation never waits for memory. Some designs also use multi cast BRAM networks so one memory block can serve several processing elements at once, improving memory efficiency and reducing bandwidth limits [29, 5, 68].

5) Loop Unrolling and Tiling: Loop unrolling means executing several loop iterations in parallel. Loop tiling divides the data into smaller blocks that fit inside local memory. Both techniques help reuse data and reduce off-chip memory access. Frameworks like Caffeine use double-buffering and tiling together to prefetch the next data block while processing the current one, increasing speed and throughput [29, 2].

6) Custom Precision and Fixed Point Arithmetic: Many FPGA accelerators use fixed-point arithmetic instead of full floating-point to save resources and power. For example, 16-bit fixed-point designs can be up to three times faster than floating-point versions. Binarized networks go even further, achieving over ten times speed improvement while using far fewer logic elements [29, 57, 2].

7) Streaming Architectures: Streaming architectures process data as a continuous flow between layers. In the fpgaConvNet framework, layers are connected by on-chip streams, and only weights are reloaded between subgraphs. This avoids full reconfiguration and makes execution much faster [29, 61]. Similarly, the FINN architecture uses small streaming blocks for each layer that send results directly to the next one, supporting real-time inference [29, 57].

VI. Related Work and Current Results

A. Surveyed FPGA Accelerators (Quantitative) and Representative Case Studies

ShiDianNao and DadianNao are both ASIC based hardware accelerators made for running neural networks very efficiently. they are not FPGAs but fixed ASIC chips. ShiDianNao runs at 1 GHz and can do 194 billion operatios per second while only using 320.10 MW. this is what makes them 60 times more energy eficient than an older chip called DianNao [7].

DaDianNao is a bigger version that connects several chips together. it performs up to 656 times faster than a GPU and uses 184 times energy [29].

Qiu et al. [46] and his team proposed a new accelerator which was based on Zynq XC7Z045 FPGA for VGG-16 neural network [29]. They used 16 bit fixed points insted of useing 32 bit floats to save FPGA resources [29]. They applied They also applied a mathematical method known as Singular Value Decomposition (SVD) What this methond does is it breaks large matrix into smaller parts so they are esay to work with, That way they were able to shrink the large fully connected layers, reducing the memory size by 85% while losing only 0.04% accuracy [29]. That way their FPGA design was able to achive 136.97 Billion operations per second at only 150 MHz clock speed only using 9.6 watts [29]. Lu et al. [37] Introduced the FLexFlow accelerator it is a newer FPGA design that focuses on dataflow flexibility, it supporting all combinations of CNN parallelism (synapse, neuron, and feature map) [29]. Unlike traditional systolic or 2D mapping architectures limited to one type of parallelism, FlexFlow dynamically adjusts data paths through a hierarchical dataflow system [29]. When tested on networks like AlexNet and VGG, the design used over 80% of the FPGA's computing blocks effectively and achieved more than 420 GOPS at 1 GHz. [29].

B. Discussion: Trends and Takeaways

All the designs are focused on improving the speed and energy efficiency of deep neural networks by reducing memory access and increasing parallel processing. most accelerators nearly 90% of their ttime to perform (CNN) operations, so they focused on optimizing this par first [29]

Designers often use data reuse and local memory buffers to reduce off-chip memory transfers, which consume both time and power [29]. Methods such as loop tiling and pipelining help keep all processing elements active and avoid idle cycles. These strategies are essential for overcoming the limited memory bandwidth of FPGAs [29].

Finally, the results show that FPGAs continue to offer excellent performance per watt compared to CPUs and GPUs. Even though they sometimes lag in total throughput, their reconfigurability, energy efficiency, and customizable precision make them ideal for embedded and edge AI applications where power and cost are critical [29].

VII. Challenges

FPGA based CNN accelerators face several challenges. These challenges come from memory size, bandwidth, compute resources, and tool flows, the key issues have benn summarized below [29].

A. Hardware Resource Limits (DSP/BRAM/LUT/FF)

One of the biggest challenges in implementing deep neural network on FPGAs is the limited amount of hardware resources such as DSP blocks, LUTs, and flip flop [56]. Large CNNs like AlexNet and VGG contail around millions of parameters and require billion of operations oer second, which is more than a FPGA can manage to fit a single chip [54, 46]. AlexNet needs about 250 MB to store its weights and 1.5 billion operations for one image, while VGG-16 requires 138 million weights and over 30 GOPS [54, 46]. Because current FPGAs do not have enough on chip memory, weights and feature maps must be stored in external DRAM and repeatedly transferred during processing. This increases latency and makes full model implementation difficult. To overcome this, designers try to share computational units, reuse logic resources, and optimize data reuse to fit as much of the model as possible into the FPGA fabric [29].

B. Memory Bandwidth Bottlenecks

Even if computation fits inside the FPGA, the system faces limitations by memory bandwidth [56]. Most CNN layers need to fetch larg ammounts of data weights, activations, and feature maps from external memory. The speed of data movment between the FPGA and DRAM is slower than the actual computation speed, because current FPGAs do not have enough on chip memory to store all the weights and featurs maps [56]. Moreover, accesing external memory consumes much more power as compaired to on chip memory [56]. To reduce this problem, several optimization methods have been proposed, such as reducing precision, reusing data on chip, and minimizing external memory access . Data reuse through buffering and tiling further decreases off chip traffic [56]. Despite these improvements, external memory bandwidth remains one of the most critical challenges in FPGA based CNN accelerators [29].

C. Power and Thermal Constraints

Power consumption is a growing concern for deep learning accelerators. GPUs, for example, deliver high throughput but consume a large amount of energy, which makes them unsuitable for low power or portable applications [28, 30, 29]. FPGAs are generally more energy efficient, but when large networks are implemented, their frequent access to off chip memory and high switching activity can still result in significant power usage and heat generation [22, 29]. This becomes especially challenging for edge devices that operate on batteries or have limited cooling. Researchers attempt to reduce this by lowering data precision, optimizing dataflow to reduce unnecessary transfers, and gating unused parts of the FPGA to save dynamic power [29].

D. Design Complexity and Exploration Time

Designing efficient FPGA accelerators for neural networks is not straightforward. Different CNN layers such as convolutional, pooling, and fully connected layers have very different computational and memory requirements. Convolutional layers are computation heavy, while fully connected layers are memory heavy [46]. Achieving the best balance for all layers requires careful design choices such as loop unrolling, tiling, parallelization, and pipelining. Furthermore, CNNs continue to grow in depth and parameter count, which increases both model complexity and implementation difficulty [29]. FPGA design tools also require long synthesis and place and route times, often taking hours for a single iteration. This makes it difficult to explore many design alternatives. To overcome this, researchers use high level synthesis tools and analytical models to guide the search for efficient architectures and to reduce development time [29].

E. Quantization and Precision Trade Offs

Reducing data precision is one of the most effective methods to save FPGA resources and power, but it comes with a trade off. Using smaller bit widths such as 8-bit or even 1-bit for binary networks reduces memory use, bandwidth, and computation cost [56, 29]. However, aggressive quantization can lead to a noticeable drop in model accuracy. Designers must find a balance between efficiency and accuracy by applying quantization aware training, pruning less important connections, or using mixed precision approaches where sensitive layers keep higher precision while others use lower precision. As noted in [46, 29], reducing precision also helps lower off chip memory bandwidth requirements and improves overall energy efficiency, but choosing the optimal quantization level remains an open research problem because it depends on both the neural network architecture and the FPGA hardware characteristics.

F. Reconfiguration Overhead

One advantage of FPGAs is that they can be reconfigured for different network layers or applications, but this flexibility also introduces overhead. Some accelerator designs use multiple bitstreams or subgraphs to implement large networks, which requires the FPGA to reload configurations when switching between layers [29]. Full reconfiguration can take a significant amount of time, reducing overall throughput. Researchers have shown that partial reconfiguration or weight reloading strategies can significantly reduce this cost, and one study reported that loading weights for a VGG16 layer was nearly 273 times faster than performing a full FPGA reconfiguration [29]. Even with these improvements, managing reconfiguration efficiently remains a challenge. Current work focuses on pipelined execution and multicontext architectures to overlap computation with reconfiguration and minimize performance loss [29].

G. Scaling to Large Models

As deep learning models continue to grow in size and complexity, scaling them on FPGAs becomes increasingly difficult. Larger networks require more storage and computational parallelism than a single FPGA can provide [29, 46]. Moreover, each layer of a CNN has different patterns of parallelism and memory access, which makes it difficult to design one hardware structure that performs optimally across all layers [29]. When the model is too large, designers often need to split it into smaller subgraphs or process it in batches, which can increase latency. Techniques such as compression, pruning, and model partitioning help fit large models into limited FPGA resources, and reducing the data bitwidth can also ease the memory and bandwidth requirements [29]. Even with these methods, scaling remains one of the most challenging open problems in FPGA-based deep learning acceleration.

H. Comparison to GPU/ASIC Limits

While FPGAs address some limitations of GPUs and ASICs, they also have their own trade-offs. GPUs are easy to program and provide massive parallelism, but they consume high power and perform best with large batch sizes, which increases latency [28, 31]. ASICs, on the other hand, offer maximum performance and efficiency but require long design cycles and cannot be modified once fabricated [29]. FPGAs provide a middle ground as they are reconfigurable, power-efficient, and capable of achieving low latency [29]. However, they still face resource, bandwidth, and design challenges that limit their scalability for very large or frequently changing neural networks. Ongoing research aims to close this gap by improving FPGA architectures, optimizing memory hierarchies, and advancing design automation tools.

VIII. Conclusion

FPGAs are becoming one of the most important platforms for accelerating neural networks because they combine high performance with low power use and flexibility. Unlike CPUs and GPUs that follow fixed architectures, an FPGA can be programmed to match the structure of each neural network layer. This allows operations like convolutions and matrix multiplications to run in parallel using DSP blocks, pipelining, and local memory buffers that reduce the need to access external memory. These features make FPGAs very suitable for real time tasks such as autonomous driving, robotics, and edge computing.

However, FPGA based accelerators still face challenges such as limited hardware resources, memory bandwidth bottlenecks, and long design times. Large models are difficult to fit into one FPGA, and reconfiguration between layers can add delay. To solve these problems, researchers are improving design tools, memory hierarchies, and using model compression and quantization to reduce size and power use.

Overall, FPGAs offer a strong middle ground between GPUs and ASICs. They are energy efficient, reprogrammable, and can achieve low latency while keeping reasonable throughput. With better high level design tools and more advanced architectures, future FPGA systems will be able to handle larger and more complex neural networks even more efficiently.

References

[1] L. Lu et al. “Two-Dimensional Winograd Algorithm on FPGA”. In: *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.* 2017, pp. 4013–4021.

[2] Y. Guan et al. “FP-DNN: An Automated Framework for Mapping Deep Neural Networks onto FPGAs”. In: *Proc. IEEE 25th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM).* 2017, pp. 152–159.

[3] H. Ali et al. “A Survey on FPGA-Based Accelerator for Machine Learning”. In: *2021 International Conference on Artificial Intelligence in Information and Communication (ICAIIC).* IEEE, 2021, pp. 145–150. DOI: 10.1109/ICAIIC51459.2021.9415204.

[4] Dario Amodei et al. “Deep Speech 2: End-To-End Speech Recognition in English and Mandarin”. In: *arXiv preprint arXiv:1512.02595* (2015). URL: https://arxiv.org/abs/1512.02595.

[5] U. Aydonat et al. “An OpenCL Deep Learning Accelerator on Arria 10”. In: *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays.* 2017, pp. 55–64.

[6] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao. “YOLOv4: Optimal Speed and Accuracy of Object Detection”. In: *arXiv preprint arXiv:2004.10934* (2020).

[7] T. Chen et al. “DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning”. In: *ACM SIGPLAN Notices.* Vol. 49. 4. 2014, pp. 269–284.

[8] T. M. Chilimbi et al. “Project Adam: Building an efficient and scalable deep learning training system”. In: *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI).* Vol. 14. taken from FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review. 2014, pp. 571–582.

[9] P. Colangelo et al. “Exploration of low numeric precision deep learning inference using Intel FPGAs”. In: *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM).* 2018, pp. 73–80.

[10] Ronan Collobert et al. "Natural Language Processing (Almost) From Scratch". In: *arXiv preprint arXiv:1103.0398* (2011). URL: https://arxiv.org/abs/1103.0398.

[11] D. Diamantopoulos and C. Hagleitner. "A system-level transprecision FPGA accelerator for BLSTM using on-chip memory reshaping". In: *2018 International Conference on Field-Programmable Technology (FPT)*. IEEE, 2018, pp. 338–341.

[12] Gregory Diamos et al. "Persistent RNNs: Stashing Recurrent Weights On-Chip". In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2016.

[13] L. Du et al. "A Reconfigurable Streaming Deep Convolutional Neural Network Accelerator for Internet of Things". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 65.1 (2018). taken from FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review, pp. 198–208. DOI: 10.1109/TCSI.2017.2705055.

[14] H. Esmaeilzadeh et al. "Neural acceleration for general-purpose approximate programs". In: *45th Annual IEEE/ACM International Symposium on Microarchitecture*. taken from FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review. 2012, pp. 449–460. DOI: 10.1109/MICRO.2012.52.

[15] C. Farabet et al. "Large-Scale FPGA-Based Convolutional Networks". In: *Scaling up Machine Learning: Parallel and Distributed Approaches*. Ed. by R. Bekkerman, M. Bilenko, and J. Langford. Cambridge, U.K.: Cambridge University Press, 2011, pp. 399–419.

[16] S. B. Goldberg et al. "Machine Learning and Natural Language Processing in Psychotherapy Research: Alliance as Example Use Case". In: *Journal of Counseling Psychology* 67.4 (2020), p. 438.

[17] Y. Gong et al. "N3H-Core: Neuron-designed neural network accelerator via FPGA-based heterogeneous computing cores". In: *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 2022, pp. 112–122.

[18] Alex Graves and Jürgen Schmidhuber. "Framewise Phoneme Classification With Bidirectional LSTM and Other Neural Network Architectures". In: *Neural Networks* (2005).

[19] K. Guo et al. "Angel-Eye: A Complete Design Flow for Mapping CNN onto Embedded FPGA". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.1 (2018). taken from FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review, pp. 35–47. DOI: 10.1109/TCAD.2017.2760518.

[20] E. Haghighat and R. Juanes. "SciANN: A Keras/TensorFlow Wrapper for Scientific Computations and Physics-Informed Deep Learning Using Artificial Neural Networks". In: *Computer Methods in Applied Mechanics and Engineering* 373 (2021), p. 113552.

[21] M. Hall and V. Betz. "From TensorFlow graphs to LUTs and wires: Automated sparse and physically aware CNN hardware generation". In: *2020 Int.*

[22] R. Hameed et al. "Understanding Sources of Inefficiency in General-Purpose Chips". In: *ACM SIGARCH Computer Architecture News* 38.3 (June 2010), pp. 37–47.

[23] S. Han et al. "EIE: Efficient Inference Engine on Compressed Deep Neural Network". In: *43rd Annual International Symposium on Computer Architecture (ISCA)*. taken from FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review. 2016, pp. 243–254. DOI: 10.1109/ISCA.2016.30.

[24] S. Han et al. "Learning Both Weights and Connections for Efficient Neural Network". In: *Advances in Neural Information Processing Systems*. Vol. 28. 2015.

[25] Awni Hannun et al. "Deep Speech: Scaling Up End-To-End Speech Recognition". In: *arXiv preprint arXiv:1412.5567* (2014). URL: https://arxiv.org/abs/1412.5567.

[26] K. He et al. "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778.

[27] M. C. Herbordt et al. "Computing Models for FPGA-Based Accelerators". In: *Computer Science and Engineering* 10.6 (2008), pp. 35–45.

[28] G. Hinton et al. "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups". In: *IEEE Signal Processing Magazine* 29.6 (2012). taken from FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review, pp. 82–97. DOI: 10.1109/MSP.2012.2205597.

[29] Asmaa Ibrahim, Ashraf Attia, and Nayera Hussein. "FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review". In: *IEEE Access* 8 (2020). used its abstract to write the introduction, pp. 134824–134849. DOI: 10.1109/ACCESS.2020.3009390.

[30] H. Irmak, D. Ziener, and N. Alachiotis. "Increasing flexibility of FPGA-based CNN accelerators with dynamic partial reconfiguration". In: *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*. 2021, pp. 306–311.

[31] Y. Jia et al. "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *Proceedings of the 22nd ACM International Conference on Multimedia*. 2014, pp. 675–678.

[32] G. G. Ko et al. "Accelerating Bayesian inference on structured graphs using parallel Gibbs sampling". In: *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. 2019, pp. 159–165.

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*. 2012.

[34] G. Lacey, G. W. Taylor, and S. Areibi. *Deep Learning on FPGAs: Past, Present, and Future*. arXiv preprint arXiv:1602.04283. 2016. URL: https://arxiv.org/abs/1602.04283.

[35] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep Learning”. In: *Nature* 521 (2015), pp. 436–444.

[36] L. Lu et al. “Evaluating Fast Algorithms for Convolutional Neural Networks on FPGAs”. In: *Proceedings of the IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 2017, pp. 101–108.

[37] W. Lu et al. “FlexFlow: A Flexible Dataflow Accelerator Architecture for Convolutional Neural Networks”. In: *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*. Feb. 2017, pp. 553–564.

[38] J. Misra and I. Saha. “Artificial Neural Networks in Hardware: A Survey of Two Decades of Progress”. In: *Neurocomputing* 74.1–3 (2010). taken from FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review, pp. 239–255. DOI: 10.1016/j.neucom.2010.03.021.

[39] A. Munshi. “The OpenCL Specification”. In: *Proceedings of the IEEE Hot Chips 21 Symposium (HCS)*. 2009, pp. 13–14.

[40] M. S. Murshed et al. “Resource-Aware On-Device Deep Learning for Supermarket Hazard Detection”. In: *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2020, pp. 871–876.

[41] H. Nakahara et al. “A lightweight YOLOv2: A binarized CNN with a parallel support vector regression for an FPGA”. In: *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 2018, pp. 31–40.

[42] O. Nomura and T. Morie. “Projection-field-type VLSI convolutional neural networks using merged/mixed analog-digital approach”. In: *Proceedings of the International Conference on Neural Information Processing*. taken from FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review. Berlin, Germany: Springer, 2007, pp. 1081–1090.

[43] A. R. Omondi and J. C. Rajapakse. *FPGA Implementations of Neural Networks*. Vol. 365. Boston, MA, USA: Springer, 2006.

[44] A. Parashar et al. “SCNN: An Accelerator for Compressed-Sparse Convolutional Neural Networks”. In: *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*. ACM, 2017, pp. 27–40. DOI: 10.1145/3079856.3080254.

[45] A. Putnam et al. “A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services”. In: *ACM SIGARCH Computer Architecture News* 42.3 (2014). taken from FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review, pp. 13–24. DOI: 10.1145/2678373.2665678.

[46] J. Qiu et al. “Going Deeper with Embedded FPGA Platform for Convolutional Neural Network”. In: *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays (FPGA)*. 2016, pp. 26–35.

[47] Z. Que et al. “A reconfigurable multithreaded accelerator for recurrent neural networks”. In: *2020 International Conference on Field-Programmable Technology (ICFPT)*. 2020, pp. 20–28.

[48] O. Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision* 115.3 (2015). taken from FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review, pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[49] V. Rybalkin and N. Wehn. “When massive GPU parallelism ain’t enough: A novel hardware architecture of 2D-LSTM neural network”. In: *FPGA 2020: ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 2020, pp. 111–121.

[50] R. Saravanan and P. Sujatha. “A State of Art Techniques on Machine Learning Algorithms: A Perspective of Supervised Learning Approaches in Data Classification”. In: *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)*. IEEE, 2018, pp. 945–949.

[51] K. Settaluri et al. “AutoCkt: Deep Reinforcement Learning of Analog Circuit Designs”. In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 490–495.

[52] B. A. Skourt, A. El Hassani, and A. Majda. “Lung CT Image Segmentation Using Deep Neural Networks”. In: *Procedia Computer Science*. Vol. 127. 2018, pp. 109–113.

[53] J. E. Stone, D. Gohara, and G. Shi. “OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems”. In: *Computer Science and Engineering* 12.3 (2010), pp. 66–73.

[54] N. Suda et al. “Throughput-Optimized OpenCL-Based FPGA Accelerator for Large-Scale Convolutional Neural Networks”. In: *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays (FPGA)*. 2016, pp. 16–25.

[55] M. Sun et al. “Hardware-friendly acceleration for deep neural networks with microstructured compression”. In: *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 2022.

[56] V. Sze et al. “Hardware for Machine Learning: Challenges and Opportunities”. In: *Proc. IEEE Custom Integrated Circuits Conf. (CICC)*. 2017, pp. 1–8.

[57] Y. Umuroglu et al. “FINN: A Framework for Fast, Scalable Binarized Neural Network Inference”. In: *Proceed-*

ings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. 2017, pp. 65–74.

[58] W. Vanderbauwhede and K. Benkrid. *High-Performance Computing Using FPGAs*. taken from FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review. New York, NY, USA: Springer, 2013.

[59] B. S. C. Varma, K. Paul, and M. Balakrishnan. *Architecture Exploration of FPGA-Based Accelerators for Bioinformatics Applications*. Singapore: Springer, 2016.

[60] A. Vasudevan, A. Anderson, and D. Gregg. "Parallel multi-channel convolution using general matrix multiplication". In: *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. taken from FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review. 2017, pp. 19–24. DOI: 10.1109/ASAP.2017.7995251.

[61] S. I. Venieris and C.-S. Bouganis. "Latency-Driven Design for FPGA-Based Convolutional Neural Networks". In: *Proc. 27th Int. Conf. Field Program. Logic Appl. (FPL)*. 2017, pp. 1–8.

[62] J. Villasenor and W. H. Mangione-Smith. "Configurable Computing". In: *Scientific American* 276.6 (1997), pp. 66–71.

[63] M. P. Véstias et al. "Hybrid dot-product calculation for convolutional neural networks in FPGA". In: *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. 2019, pp. 350–353.

[64] H. M. Waidyasooriya, M. Hariyama, and K. Uchiyama. *Design of FPGA-Based Computing Systems with OpenCL*. Cham, Switzerland: Springer, 2018.

[65] S. Vrudhula Y. Ma Y. Cao and J.-S. Seo. "Optimizing Loop Operation and Dataflow in FPGA Acceleration of Deep Convolutional Neural Networks". In: *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays (FPGA)*. 2017, pp. 45–54.

[66] A. Yazdanbakhsh et al. "Neural acceleration for GPU throughput processors". In: *Proceedings of the 48th International Symposium on Microarchitecture (MICRO)*. taken from FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review. 2015, pp. 482–493.

[67] M. Yingchang and Q. Liu. "M4BRAM: Mixed-precision matrix-matrix multiplication in FPGA block RAMs". In: *2023 International Conference on Field Programmable Technology (ICFPT)*. 2023.

[68] C. Zhang and V. Prasanna. "Frequency Domain Acceleration of Convolutional Neural Networks on CPU–FPGA Systems". In: *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays* (2017), pp. 35–44.

[69] H. Zhou et al. "Enabling Low Latency Edge Inference of Deep Neural Networks on FPGAs". In: *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2019, pp. 1–8.

List of Figures