Bulletin of Electrical Engineering and Informatics
Vol. 13, No. 6, December 2024, pp. 4008~4017
ISSN: 2302-9285, DOI: 10.11591/eei.v13i6.7683

Implementation of deep learning models in FPGA development board for recognition accuracy enhancement

Salah Ayad Jassim, Ibrahim Khider
Department of Electronics, College of Engineering, Science and Technology of Sudan University, Khartoum, Sudan

Article Info

Article history:

Received Oct 6, 2023
Revised Apr 1, 2024
Accepted May 17, 2024

Keywords:

Computational cost
Deep learning
Field programmable gate array
Performance
Training

ABSTRACT

Deep learning (DL) model performance is intricately tied to the quality of training, influenced by several parameters. Of these, the computing unit employed significantly impacts training efficiency. Traditional setups use central processing units (CPUs) or graphics processing units (GPUs) for DL training. This paper proposes an alternative using field programmable gate arrays (FPGAs) for DL training, leveraging their customizable and parallelizable architecture. FPGA programming allows for tailored circuit designs, optimizing DL training requirements and enabling efficient parallel processing. The use of FPGAs in DL training has garnered attention for their potential in achieving high computational throughput and energy efficiency, attributed to advantages like low latency, high bandwidth, and reconfigurability. By exploiting FPGA parallel processing capabilities, faster training times and the potential for larger, more complex DL models are feasible. This paper provides an overview of state-of-the-art techniques for FPGA-based DL model training, discussing challenges such as hardware architecture design, memory management, and algorithm optimization. Additionally, various FPGA-based DL frameworks and libraries facilitating DL model development and deployment on FPGAs are explored.

Corresponding Author:

Salah Ayad Jassim
Department of Electronics, College of Engineering, Science and Technology of Sudan University
Khartoum, Sudan
Email: salahayadvip@gmail.com

1. INTRODUCTION

Wide advancement on deep learning (DL) technology is motivated variety of applications for adopting this technology. Thanks to open source software that enabled periodic modification and enhancements of the existing DL libraries. Thus, the focal of existing researches are about performance enhancement in order to support the reliability of the DL applications adopted by major human related need such as health care fields [1].

Changing the number of iterations, batch size, number of filters within the layers and the impact of those configuration variation is measured and recorded. In this section, field programmable gate arrays (FPGAs) is used as alternative simulation environment where DL models are attempts for training and testing stages. FPGA is designed to provide flexible programmable environments that meet the need of designers in various applications. By help of parallel programming, it is expected to achieve more flexible training (problem learning) [2].

In this chapter, data preprocessing and classification using lasagene model over Python productivity for Zynq (PYNQ) FPGA development board is discussed. Two intelligent models were employed, each undergoing adjustments to their layer structures, and subsequently applied to the classification of modulations. Evaluation of the classification performance for both models utilized identical performance metrics, including accuracy, mean square error, and processing time. Xilinx has developed two development boards, the PYNQ-z1 and PYNQ-Z2, aimed at supporting the system-on-chip (SoC) paradigm. Leveraging the processing power integrated into Zynq boards and the programming flexibility of Python, these boards enable the implementation of robust system designs without the need for circuit diagrams or the utilization of a system-level programming language like hardware description language (HDL) [3].

The utilization of application programming interface (API) technology facilitates the interaction among hardware applications via software means. The incorporation of API technology on PYNQ development boards enables the manipulation of FPGA chip logic gates through Python code. Furthermore, PYNQ development boards integrate a multitude of input and output devices including HDMI, USB, microphone input, Arduino ports, Pmod headers, and a Raspberry Pi interface. These boards are furnished with user LEDs for status indication and sliding switches for control support [4].

2. RELATED WORK

Using FPGAs in DL-based modulation recognition has gained significant attention in recent years. Modulation recognition is a crucial task in wireless communication systems, as it involves identifying the modulation scheme used by a received signal. DL algorithms, such as convolutional neural networks (CNNs), have demonstrated remarkable performance in modulation recognition tasks. However, the computational complexity of DL models poses challenges for real-time implementation on conventional computing platforms. FPGAs offer a promising solution by providing parallel processing capabilities and hardware customization to accelerate DL algorithms [5], [6].

One key advantage of using FPGAs for DL-based modulation recognition is their ability to achieve high throughput and low latency. FPGAs can be programmed to perform computations in parallel, allowing for efficient processing of large datasets in real-time. This is particularly important in scenarios where high-speed signal processing is required, such as in wireless communication systems with multiple incoming signals. The parallel architecture of FPGAs enables the acceleration of DL algorithms, leading to faster, and more efficient modulation recognition [7], [8].

Moreover, FPGAs offer the flexibility to customize hardware architectures specifically tailored for DL tasks. By designing specialized processing units and memory structures, FPGAs can optimize the execution of DL models and reduce resource utilization. This allows for the efficient deployment of DL algorithms on FPGA platforms, achieving higher performance and energy efficiency compared to general-purpose processors. The ability to tailor the hardware architecture to the requirements of modulation recognition tasks contributes to improved accuracy and reduced inference time [9], [10].

Another advantage of using FPGAs is their suitability for deployment in resource-constrained environments. FPGAs provide a compact and power-efficient solution for real-time modulation recognition applications, making them suitable for implementation in embedded systems and edge devices. This is particularly relevant in wireless communication scenarios where low-power and small form-factor devices are desired. FPGAs enable the deployment of DL-based modulation recognition models in these constrained environments, expanding the applicability of such models to various practical applications [11], [12].

In summary, utilizing FPGAs in DL-based modulation recognition offers several advantages, including high throughput, low latency, hardware customization, and suitability for resource-constrained environments. These benefits enable real-time processing, improved accuracy, energy efficiency, and deployment in diverse wireless communication systems. The integration of FPGA technology with DL algorithms paves the way for efficient and effective modulation recognition in various practical applications. In Table 1 (in Appendix) [13]-[22] list a comparison for different models of FPGA depemging on memory utilization, throughput, complexity, and so on as mentioned in it.

3. DEEP LEARNING LIBRARIES AND CAFFE PROJECT

DL libraries in PYNQ FPGA provide a convenient and efficient platform for implementing DL models on FPGAs. PYNQ, which stands for python productivity for Zynq, is an open-source framework that combines the power of Python programming language and the flexibility of Zynq SoC devices. It enables software developers and researchers to leverage the capabilities of FPGAs for accelerating DL workloads. Several DL libraries have been integrated into the PYNQ framework, offering a range of functionalities and ease of use see in Figure 1 [23].

One widely used DL library in PYNQ is TensorFlow. TensorFlow is a popular open-source framework for developing and training deep neural networks (DNNs). It provides a comprehensive set of tools and APIs for building and deploying machine learning models. The integration of TensorFlow into PYNQ allows users to leverage the extensive ecosystem of TensorFlow, including pre-trained models, optimization techniques, and deployment options. TensorFlow enables efficient execution of DL models on

FPGAs through the PYNQ interface, unlocking the potential for high-performance inference and acceleration [24].

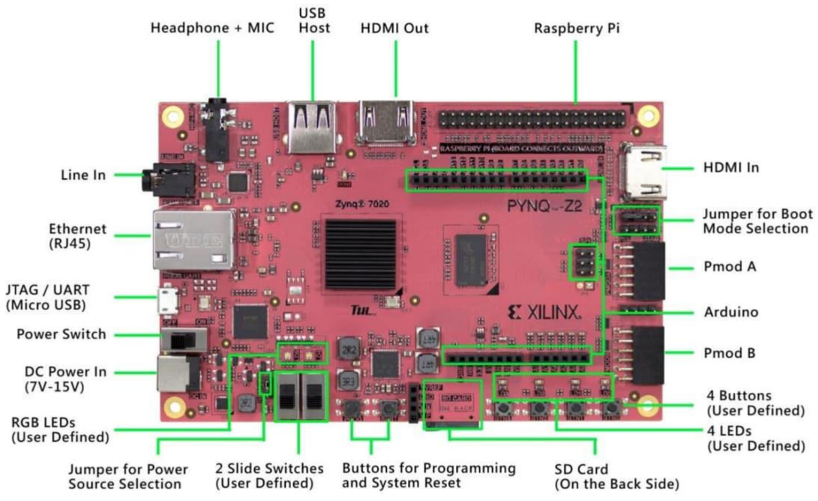


Figure 1. PYNQ-Z2 FPGA development board structure

Another DL library available in PYNQ is PyTorch. PyTorch is a dynamic DL framework that emphasizes flexibility and ease of use. It offers a dynamic computational graph, allowing for easy model development and debugging. PyTorch provides a wide range of functionalities for training and deploying DL models. Its integration with PYNQ enables users to take advantage of PyTorch's capabilities while leveraging the FPGA acceleration provided by PYNQ. This combination allows for seamless development and deployment of DL models on FPGAs [23], [24].

In addition to TensorFlow and PyTorch, PYNQ also supports other DL libraries such as Keras, Caffe, and MXNet. These libraries offer different features and capabilities, catering to diverse DL requirements. Users can choose the library that best suits their needs and preferences while utilizing the benefits of FPGA acceleration through the PYNQ framework. The availability of multiple DL libraries in PYNQ expands the range of options for developers and researchers, facilitating the development and deployment of DL models on FPGAs [24].

The integration of DL libraries into the PYNQ framework provides a user-friendly and accessible environment for FPGA-based DL. By combining the power of FPGAs with the simplicity of DL libraries, PYNQ enables rapid prototyping, efficient deployment, and accelerated performance of DL models. It offers a seamless interface between software and hardware, allowing users to harness the full potential of FPGAs for DL tasks [23], [24].

Convolutional architecture for fast feature embedding (CAFFE) presents multimedia researchers and professionals with a robust and adaptable framework for deploying state-of-the-art DL algorithms, accompanied by a repository of benchmark models. This framework, distributed under the BSD license, constitutes a C++ library supplemented with Python and MATLAB bindings, facilitating the efficient training and deployment of general-purpose CNNs and other deep models on standard hardware architectures [25]. Caffe is designed to address the computational requirements of industry and internet-scale media applications through CUDA GPU computation, achieving the processing of over 40 million images daily on a single K40 or Titan GPU (approximately 2.5 milliseconds per image). Its architecture, which separates model representation from implementation, fosters experimentation and enables smooth platform transitions. This characteristic facilitates ease of development and deployment across a spectrum of environments, ranging from prototyping machines to cloud-based infrastructures [26]. Caffe is meticulously maintained and developed under the auspices of the berkeley vision and learning center (BVLC), with the invaluable support of an engaged and dynamic community of contributors on the GitHub platform. This framework serves as a cornerstone for numerous research initiatives, as well as for the deployment of large-scale industrial applications and the development of innovative prototypes in areas encompassing vision, speech, and multimedia studies [27].

4. MODEL IMPLEMENTATION

The growing need for effective and high-performance inference across a range of applications has drawn a lot of attention to the acceleration of DL models on FPGAs. FPGAs are ideally suited for the computational demands of DNNs because they provide parallelism and configurability. An overview of the steps involved in training DL models on FPGAs is provided in this abstract. Large datasets are often used in an iterative manner to optimise model parameters for DL models during training. The deployment of specialised hardware accelerators suited to particular neural network architectures is made possible by the flexibility of FPGAs. Our method entails translating the DL model that has been trained into a HDL representation that can be implemented on an FPGA.

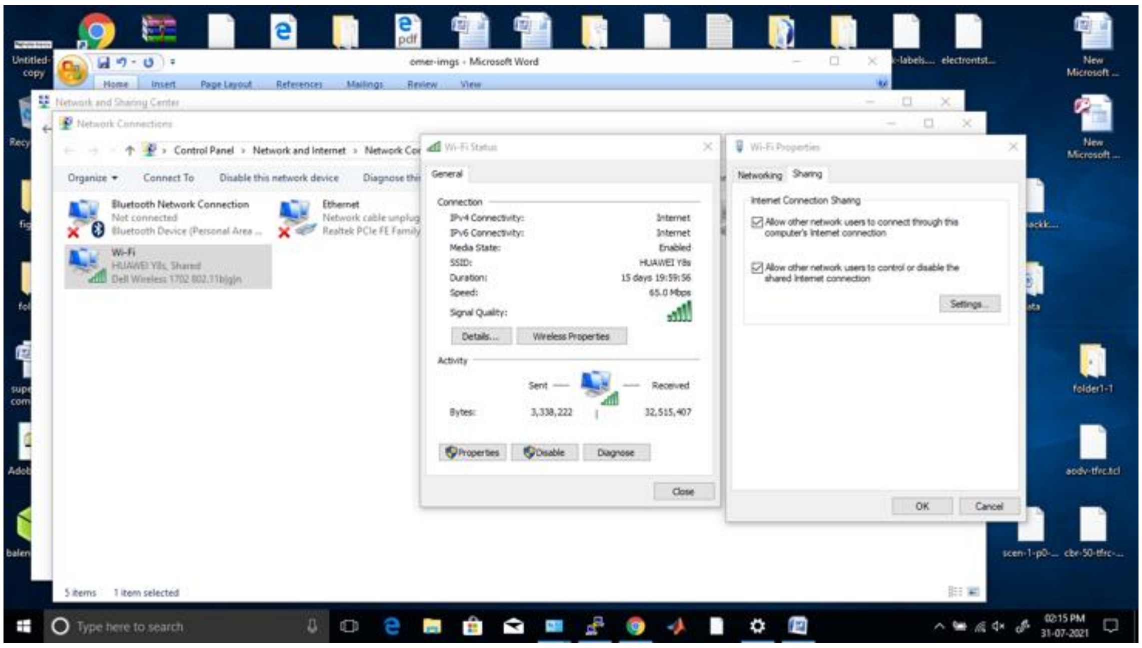
To fully utilise the parallel processing capabilities of FPGAs, we investigate methods for optimising and parallelizing convolution and matrix multiplication—two important neural network operations. During training, methods like quantization and pruning are also taken into consideration in an effort to lower resource requirements without sacrificing model accuracy. We also talk about the difficulties in training DL models on FPGAs, such as the necessity of effective memory management and balancing the use of resources against computational accuracy. Our methodology aims to attain a high-performance and resource-efficient implementation by striking a balance between these factors. Popular DL frameworks and FPGA development tools are used in experiments to show the viability and efficiency of our suggested methodology. The findings demonstrate the possibility of using FPGAs to train DL models more quickly, giving them a competitive edge over conventional GPU-based training.

In summary, this works highlights the difficulties and optimisation strategies associated with training DL models on FPGAs. By using the parallelism and reconfigurability of FPGAs, the suggested methodology seeks to expedite the DL model training process and aid in the creation of scalable and effective neural network solutions. The accelerated training of DL models on FPGAs is the main focus of this research. Inspired by FPGAs' reconfigurability and parallel processing power, we propose a methodology that builds custom hardware accelerators for critical operations and converts trained models into HDL representations. Strategies like quantization and pruning are investigated to minimise resource needs, and effective memory control and considerations for both on-chip and off-chip memory utilisation are essential for optimisation. Our experiments using popular DL frameworks and FPGA development tools demonstrate the potential of FPGA-based training in terms of improved speed, resource utilisation, and power efficiency, despite obstacles related to limited on-chip resources and design complexity. By providing a balance between computational precision and model accuracy, the methodology aids in the creation of scalable and effective neural network solutions for FPGA platforms.

4.1. Interface with computer

It is noteworthy to mention that the image flashed onto the SD card offers a Linux-based development environment conducive to the installation of Python and requisite DL libraries essential for project execution. Notably, developers utilizing the PYNQ-Z2 board often encounter a significant challenge pertaining to the installation of DL libraries, wherein error messages, predominantly indicating "source cannot be found," are prevalent. To address such recurring errors, the PYNQ-Z2 development board is commonly interfaced with terminal emulator software such as Putty, facilitating effective troubleshooting and resolution [28]. Upon reaching this stage, it is imperative to fulfill three principal requirements:

– The procedure entails inserting the SD card into the development board and establishing connections between the board and the computer via both a micro-USB cable and an ethernet cable. Power is supplied to the development board through the micro-USB cable, drawing +5 V DC from the computer, while the ethernet connection facilitates data exchange between the FPGA network card and the computer network card. Activation of the board is achieved using the ON-OFF sliding switch, with careful attention to ensuring that the bin connector on the board is set for SD card boot, as delineated in Figure 2. This configuration enables the FPGA to initiate the system flashed on the SD card.
– Before proceeding with subsequent steps, it is imperative to establish an internet connection for the FPGA. This can be accomplished through internet sharing from the PC's network card. Accessing the "network and sharing center" and selecting the main network card (local area network) enables the configuration of internet sharing. It is important to note the absence of any assigned IP addressing at this stage. Subsequent to enabling internet sharing via the local area network card, the computer will autonomously allocate an IP address to the FPGA board, typically designated as 192.168.132.1.


Network Connections
Control Panel > Network and Internet > Network Connections
Bluetooth Network Connection
Not connected
Bluetooth Device (Personal Area ...
Ethernet
Network cable unplugged
Realtek PCIe FE Family
Wi-Fi
HUAWEI Y8s, Shared
Dell Wireless 1702 802.11b|g|n

Wi-Fi Status — General
Connection
IPv4 Connectivity: Internet
IPv6 Connectivity: Internet
Media State: Enabled
SSID: HUAWEI Y8s
Duration: 15 days 19:59:56
Speed: 65.0 Mbps
Activity
Bytes: Sent 3,338,222 Received 32,515,407

Wi-Fi Properties — Sharing
Internet Connection Sharing
[✓] Allow other network users to connect through this computer's Internet connection
[✓] Allow other network users to control or disable the shared Internet connection

Figure 2. Network configuration page

– The setup of putty is pivotal to gain access to the operating system of the FPGA, which has been equipped with internet capabilities, facilitating the adjustment of Python configurations. Two pivotal factors necessitate careful attention to ensure the successful access to the terminal of the PYNQ [27], [28].
  a. To determine the hostname and clock size, it is necessary to access the "Device Manager" to identify all USB ports, thereby discerning the specific name of the USB port. Following this, the Putty settings should be configured to facilitate Telnet transfer.
  b. The clock size must be configured to 115200 Hz. This setting is illustrated in Figure 3, depicting the Putty front end.

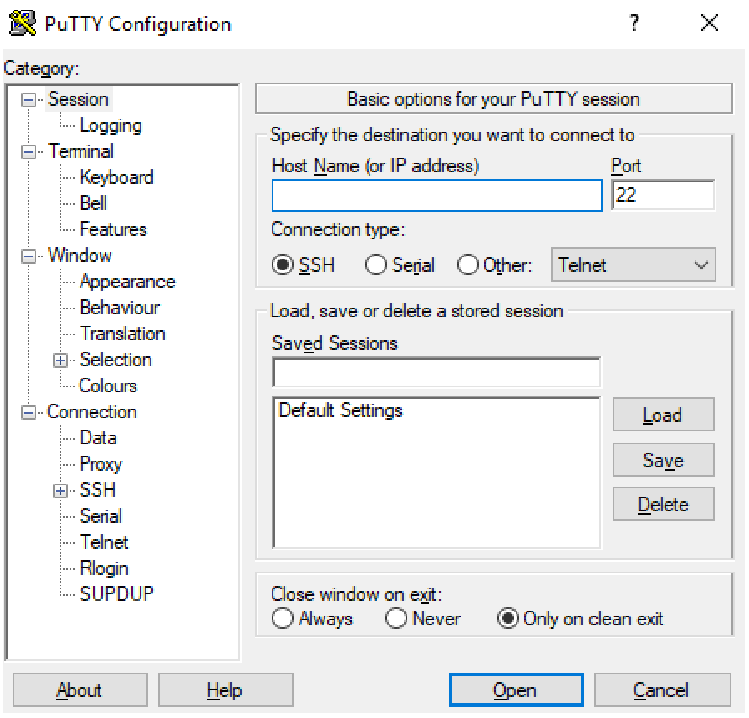

PuTTY Configuration
Basic options for your PuTTY session
Specify the destination you want to connect to
Host Name (or IP address):
Port: 22
Connection type: [✓ SSH] Serial Other: Telnet
Load, save or delete a stored session
Saved Sessions: Default Settings
Close window on exit: Always Never [✓ Only on clean exit]

Figure 3. Putty software configuration page

Table 2 show the parameters that are used for tuning of derives adaptive moment estimation (ADAM). This optimization algorithm is a stochastic gradient descent extension that updates network weights during training.

Table 2. Parameters [29]

Parameter	Details
Training model	ADAM
Number of epochs	20
Batch size	1

4.2. Model 1

After uploading database file into notebook, here all data rows need to read and stored on separated array; the same procedure of the simulation stage is used over here unless the rows dimensions with is (50×50) points are resampled to match the DL model. A new dimension similar to (1×50×50) is made. Table 3 is demonstrating the model structure. This model is trained using ADAM algorithm with 20 epochs and batch size of 20 samples as illustrated. Thereafter, model is being trained for error minimization at the detection results, however, the results are given in the Table 4.

Table 3. The first proposed model of FPGA based modulation classification detection

Layer	Configurations
Input Layer	shape=(None, 1,50,50)

Conv2DLayer	num_filters=32
	filter_size=(3, 3)

Gain layer	ReLU
MaxPool2DLayer	size=(2, 2))

Conv2DLayer	num_filters=64
	filter_size=(3, 3)

Gain layer	ReLU
MaxPool2DLayer	pool_size=(2, 2)
DenseLayer	num_units=128
Gain layer	ReLU
DropoutLayer	Probability= 0.5
DenseLayer	num_units=3
Gain Layer	softmax

Table 4. Epoch wise results (mse and time computation) for the first proposed model

Epoch	Train error	Time
1/20	0.404127	39.87
2/20	0.396219	39.84
3/20	0.393183	39.73
4/20	0.390402	39.81
5/20	0.387714	39.85
6/20	0.385068	39.87
7/20	0.382966	39.76
8/20	0.383447	39.83
9/20	0.377513	39.87
10/20	0.374564	39.9
11/20	0.371899	39.74
12/20	0.369401	39.87
13/20	0.366815	39.88
14/20	0.364147	39.76
15/20	0.361642	39.86
16/20	0.359146	39.87
17/20	0.357675	39.87
18/20	0.354271	39.73
19/20	0.351365	39.87
20/20	0.348738	39.84

4.3. Model 2

With the aim of reducing prediction errors and minimizing processing time, Model 2 has been constructed with three convolutional layers. While the structure of this model bears similarity to that proposed in the CPU (CNN) model, detailed configurations for Model 2 are provided in Table 5. Results obtained from model 2 are detailed in Table 6, knowingly, model 2 is trained for 15 epochs and the batch size was 15 samples per epoch as given in Table 7.

Table 5. State of the art model based on FPGA environments (model 2)

Layer	Configurations
InputLayer	shape=(None, 1,50,50)

Conv2DLayer	num_filters=5
	filter_size=(2, 2)

Gain layer	ReLU
MaxPool2DLayer	size=(2, 2)

Conv2DLayer	num_filters=5
	filter_size=(2, 2)

Gain layer	ReLU
MaxPool2DLayer	pool_size=(2, 2)

Conv2DLayer	num_filters=5
	filter_size=(2, 2)

Gain layer	ReLU
MaxPool2DLayer	pool_size=(2, 2)
Flatten layer	---
Gain layer	ReLU
DenseLayer	num_units=5
Gain Layer	ReLU
DenseLayer	num_units=5
Gain layer	Softmax

Table 6. Epoch wise results (mse and time computation) for the second (state of the art) proposed model

Epoch	Train error	Time/sec
1/15	0.868096	16
2/15	0.594069.	16.33
3/15	0.558680.	15.87
4/15	0.528674.	15.93
5/15	0.499050.	16.65
6/15	0.453302.	16.01
7/15	0.433587.	15.87
8/15	0.423658.	16.04
9/15	0.417319.	16.29
10/15	0.411838.	15.87
11/15	0.406916.	15.91
12/15	0.402118.	16.38
13/15	0.397430.	15.89
14/15	0.393335.	15.88
15/15	0.389542.	16.07

Table 7. Configuration of second DL model [29]

Parameter	Details
Training model	ADAM
Number of epochs	15
Batch size	15

5. CONCLUSION

FPGAs is explained in this paper as powerful alternative for training of DL algorithms. It provides enhanced training accuracy in less training time. The FPGA works as single task computer where all the processing power and random access memory can be used to serve one particular application only. That differs from the conventional computers as many other applications are sharing the random access memory and the processor at same time. So, in FPGA, the computational process are dedicated for single task. This paper explained the technique of how implement DL project with FPGA board and how to configure it to enhance the training accuracy.

APPENDIX

Table 1. Comparison of different FPGA models using different networks

Paper	Method	Pros	Cons	Critisisim
[13]	Custom FPGA accelerators for CNNs	Achieves high throughput	Limited to specific CNN architectures	Lack of exploration into other DL models
	Memory optimization for large-scale models	Efficient memory utilization	Increased design complexity	Limited discussion on real-world dataset impact
	Quantization for reduced precision	Low power consumption	Limited support for dynamic precision	Comparative analysis lacks diverse FPGA platforms
[14]	HLS-based deployment for DL models	Ease of model integration	Lower performance compared to custom designs	Limited exploration of FPGA-specific optimization
	Parallelism exploration in FPGA architectures	Scalability for various network sizes	Requires high-level FPGA programming skills	Lack of in-depth analysis on resource utilization
	Pruning techniques for resource-efficient models	Reduced resource requirements	Challenges in maintaining model accuracy	Limited evaluation on real-time applications
[15]	FPGA-friendly model conversion using high-level tools	Flexibility in model deployment	Limited support for certain layer types	Insufficient exploration of parallelism in FPGA
	Exploration of parallel processing on FPGAs	Improved training speed	Complexity in synchronizing parallel tasks	Limited discussion on FPGA-specific optimization strategies
	Quantization and pruning for resource efficiency	Balanced precision-resource trade-off	Challenges in maintaining model accuracy	Lack of comprehensive comparison with GPU-based training
[16]	FPGA-based hardware acceleration for RNNs	Efficient implementation of RNN layers	Limited scalability for larger networks	Lack of exploration into other recurrent architectures
	Memory optimization strategies for recurrent models	Improved memory bandwidth utilization	Higher FPGA resource utilization	Limited discussion on long-term dependencies in RNNs
	Pruning techniques tailored for recurrent networks	Reduced parameter counts for faster training	Impact on capturing long-term dependencies	Limited evaluation on real-world sequential data
[17]	FPGA-based deployment for transformer architectures	High parallelism for attention mechanisms	Limited support for very large models	Insufficient exploration of multi-head attention structures
	Memory optimization for transformer-based models	Efficient utilization of attention layers	Increased complexity in hardware design	Limited analysis on scalability for language models
	Quantization and pruning in transformer models	Reduced resource requirements	Challenges in maintaining model accuracy	Limited comparative analysis with GPU-based transformer training
[18]	FPGA-based sparse neural networks implementation	Efficient inference with sparse connectivity	Limited support for dense models	Lack of exploration into different sparsity-inducing techniques
	Quantization and pruning for sparse models	Reduced resource requirements	Challenges in training sparse networks	Limited evaluation on real-world sparse datasets
	Memory optimization for sparse neural networks	Efficient memory utilization	Increased complexity in sparse training	Lack of comparative analysis with other sparse neural network implementations
[19]	FPGA-based implementation of capsule networks	Efficient routing in capsule layers	Limited scalability for larger networks	Lack of exploration into dynamic routing mechanisms
	Memory optimization for capsule networks	Efficient memory utilization	Increased design complexity	Limited analysis on the impact of dynamic routing on inference
	Quantization and pruning strategies for capsules	Reduced resource requirements	Challenges in maintaining model accuracy	Lack of comprehensive comparison with GPU-based capsule training
[20]	FPGA-based acceleration for graph neural networks	Efficient implementation of graph layers	Limited support for very large graphs	Insufficient exploration of different graph neural network architectures
	Quantization and pruning for graph neural networks	Reduced resource requirements	Challenges in capturing complex structures	Limited evaluation on real-world graph datasets
	Memory optimization for graph-based models	Efficient memory utilization	Increased complexity in hardware design	Lack of comprehensive comparison with GPU-based GNN training

Table 1. Comparison of different FPGA models using different networks *(continued)*

Paper	Method	Pros	Cons	Critisisim
[21]	FPGA-based training of generative adversarial networks	Parallelized optimization for GANs	Limited support for very large models	Lack of exploration into different GAN architectures
	Quantization and pruning strategies for GANs	Reduced resource requirements	Challenges in training complex GANs	Limited evaluation on diverse image generation tasks
	Memory optimization for GANs	Efficient memory utilization	Increased complexity in hardware design	Lack of comprehensive comparison with GPU-based GAN training
[22]	FPGA-based deployment of deep reinforcement learning	Efficient implementation of RL algorithms	Limited scalability for complex environments	Lack of exploration into different RL algorithms
	Memory optimization for reinforcement learning	Efficient memory utilization	Increased design complexity	Limited analysis on the impact of FPGA acceleration on RL training
	Quantization and pruning for reinforcement learning	Reduced resource requirements	Challenges in training complex RL models	Limited evaluation on real-world RL environments

REFERENCES

[1] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello, "Hardware accelerated convolutional neural networks for synthetic vision systems," in *ISCAS 2010 - 2010 IEEE International Symposium on Circuits and Systems: Nano-Bio Circuit Fabrics and Systems*, May 2010, pp. 257–260, doi: 10.1109/ISCAS.2010.5537908.

[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2323, 1998, doi: 10.1109/5.726791.

[3] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," *arXiv*, 2014, doi: 10.48550/arXiv.1404.5997.

[4] S. Chetlur *et al.*, "cuDNN: Efficient Primitives for Deep Learning," *arXiv*, Oct. 2014, doi: 10.48550/arXiv.1410.0759.

[5] B. Jdid, K. Hassan, I. Dayoub, W. H. Lim, and M. Mokayef, "Machine Learning Based Automatic Modulation Recognition for Wireless Communications: A Comprehensive Survey," *IEEE Access*, vol. 9, pp. 57851–57873, 2021, doi: 10.1109/ACCESS.2021.3071801.

[6] M. lotfy, M. Essai, and H. Atallah, "Automatic Modulation Classification: Convolutional Deep Learning Neural Networks Approaches," *SVU-International Journal of Engineering Sciences and Applications (SVU-IJESA)*, vol. 4, no. 1, pp. 48–54, 2023, doi: 10.21608/svusrc.2022.162662.1076.

[7] W. Xiao, Z. Luo, and Q. Hu, "A review of research on signal modulation recognition based on deep learning," *Electronics*, vol. 11, no. 17, p. 2764, 2022.

[8] R. Zhou, F. Liu, and C. W. Gravelle, "Deep learning for modulation recognition: A survey with a demonstration," *IEEE Access*, vol. 8, pp. 67366–67376, 2020.

[9] M. Isik, K. Inadagbo, and H. Aktas, "Design optimization for high-performance computing using FPGA," *arXiv*, 2023, doi: 10.48550/arXiv.2304.12474

[10] A. Shawahna, S. M. Sait, and A. El-Maleh, "FPGA-based accelerators of deep learning networks for learning and classification: A review," *ieee Access*, vol. 7, pp. 7823–7859, 2018.

[11] K. P. Seng, P. J. Lee, and L. M. Ang, "Embedded intelligence on fpga: Survey, applications and challenges," *Electronics*, vol. 10, no. 8, pp. 1–33, 2021, doi: 10.3390/electronics10080895.

[12] H. Wu, "Feature-Weighted Naive Bayesian Classifier for Wireless Network Intrusion Detection," *Security and Communication Networks*, vol. 2024, pp. 1–13, 2024, doi: 10.1155/2024/7065482.

[13] Y. Ma, N. Suda, Y. Cao, S. Vrudhula, and J. sun Seo, "ALAMO: FPGA acceleration of deep learning algorithms with a modularized RTL compiler," *Integration*, vol. 62, pp. 14–23, 2018, doi: 10.1016/j.vlsi.2017.12.009.

[14] S. S. C and G. R, "Onboard target detection in hyperspectral image based on deep learning with FPGA implementation," *Microprocessors and Microsystems*, vol. 85, 2021, doi: 10.1016/j.micpro.2021.104313.

[15] A. G. Blaiech, K. Ben Khalifa, C. Valderrama, M. A. C. Fernandes, and M. H. Bedoui, "A Survey and Taxonomy of FPGA-based Deep Learning Accelerators," *Journal of Systems Architecture*, vol. 98, pp. 331–345, 2019, doi: 10.1016/j.sysarc.2019.01.007.

[16] V. B. K. L. Aruna, E. Chitra, and M. Padmaja, "Accelerating deep convolutional neural network on FPGA for ECG signal classification," *Microprocessors and Microsystems*, vol. 103, 2023, doi: 10.1016/j.micpro.2023.104939.

[17] H. Sriraman and A. Ravikumar, "Customized FPGA Design and Analysis of Soft-Core Processor for DNN," *Procedia Computer Science*, vol. 218, pp. 469–478, 2022, doi: 10.1016/j.procs.2023.01.029.

[18] K. Elsaid, M. W. El-Kharashi, and M. Safar, "An optimized FPGA architecture for machine learning applications," *AEU - International Journal of Electronics and Communications*, vol. 174, 2024, doi: 10.1016/j.aeue.2023.155011.

[19] Y. Jin, Q. Wan, X. Wu, X. Fu, and J. Chen, "FPGA-accelerated deep neural network for real-time inversion of geosteering data," *Geoenergy Science and Engineering*, vol. 224, 2023, doi: 10.1016/j.geoen.2023.211610.

[20] M. L. Zhu and D. Y. Ge, "Image quality assessment based on deep learning with FPGA implementation," *Signal Processing: Image Communication*, vol. 83, 2020, doi: 10.1016/j.image.2020.115780.

[21] M. Astrain, M. Ruiz, A. Carpeño, S. Esquembri, and D. Rivilla, "Development of deep learning applications in FPGA-based fusion diagnostics using IRIO-OpenCL and NDS," *Fusion Engineering and Design*, vol. 168, 2021, doi: 10.1016/j.fusengdes.2021.112393.

[22] T. Kowsalya, "Area and power efficient pipelined hybrid merged adders for customized deep learning framework for FPGA implementation," *Microprocessors and Microsystems*, vol. 72, 2020, doi: 10.1016/j.micpro.2019.102906.

[23] M. Dhouibi, A. K. B. Salem, and S. B. Saoud, "CNN for object recognition implementation on FPGA using PYNQ framework," in *2020 IEEE Eighth International Conference on Communications and Networking (ComNet)*, IEEE, 2020, pp. 1–6.

[24] C. Heinz, J. Hofmann, J. Korinth, L. Sommer, L. Weber, and A. Koch, "The TaPaSCo Open-Source Toolflow: for the Automated Composition of Task-Based Parallel Reconfigurable Computing Systems," *Journal of Signal Processing Systems*, vol. 93, pp.

545–563, 2021.
[25] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie, and X. Zhou, "DLAU: A scalable deep learning accelerator unit on FPGA," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 3, pp. 513–517, 2017, doi: 10.1109/TCAD.2016.2587683.
[26] A. X. M. Chang, B. Martini, and E. Culurciello, "Recurrent Neural Networks Hardware Implementation on FPGA," *arXiv*, 2015, doi: 10.48550/arXiv.1511.05552.
[27] Q. Xiao, Y. Liang, L. Lu, S. Yan, and Y. W. Tai, "Exploring Heterogeneous Algorithms for Accelerating Deep Convolutional Neural Networks on FPGAs," in *Proceedings - Design Automation Conference*, Jun. 2017, vol. Part 128280, pp. 1–6, doi: 10.1145/3061639.3062244.
[28] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer CNN accelerators," in *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, Oct. 2016, vol. 2016-December, pp. 1–12, doi: 10.1109/MICRO.2016.7783725.
[29] I. Kandel and M. Castelli, "The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset," *ICT Express*, vol. 6, no. 4, pp. 312–315, 2020, doi: 10.1016/j.icte.2020.04.010.

BIOGRAPHIES OF AUTHORS

Salah Ayad Jassim was born in Ramadi, Iraq, in 1990. He received a B.S. degree from Al-Anbar University, Department of Electrical, Ramadi in 2013 and an M.S. degree from the University of Technology, Baghdad, in 2016. He is currently pursuing a Ph.D. degree with the Department of Electrical and Electronic Engineering, Sudan University for science and technology. He can be contacted at email: salahayadvip@gmail.com.

Ibrahim Khider was born in Sudan, Wadshommo-Elhasaheisa in 1974. He received a B.S. degree from Sudan University of Science and Technology in Electronics Engineering in 1999, M.Sc. in Communication Engineering from University of karrary, Sudan in 2002 and the Ph.D., communication engineering and Information Systems from Huazhong University of Science and Technology, China in 2008. He is interested with mobile, wireless and wired distributed systems and networking, digital signal processing and data communication, cooperative wireless networks, wireless mesh networks and cognitive radio, vehicular mobile Ad hoc NETworks, and mobility model's design. Wireless system based IoT and visible light communication and optical fiber. He can be contacted at email: ibrahim_khider@hotmail.com.