FPGA Implemenetation of Radar-Based Hand Gesture Recognition: HDL and HLS

1st Rishit Mane
International Institute of
Information Technology-Bangalore
Bengaluru, India
rishit.mane@iiitb.ac.in

2nd Himanshu Khatri
International Institute of
Information Technology-Bangalore
Bengaluru, India
himanshu.khatri@iiitb.ac.in

3rd Prof. Rituparna Choudhury
International Institute of
Information Technology-Bangalore
Bengaluru, India
rituparna.choudhury@iiitb.ac.in

***Abstract*—This paper presents a hardware-aware study of a compact depthwise separable convolutional neural network for radar-based hand gesture recognition. The same trained network is implemented using two FPGA design flows: (i) a fully modular, handcrafted SystemVerilog accelerator and (ii) a high-level synthesis route using the Xilinx Vitis AI DPU. Both implementations are evaluated on the same Xilinx platform to compare performance, resource utilization, and power consumption. The results demonstrate clear trade-offs: the handcrafted HDL design achieves low resource usage and power with competitive accuracy, while the HLS flow provides higher throughput and faster deployment.**

***Index Terms*—FPGA acceleration, depthwise separable CNN, radar gesture recognition, HDL,HLS.**

I. Introduction

Convolutional neural networks (CNNs) are widely used for perceptual tasks, but their deployment on embedded platforms is limited by compute, memory, and energy constraints. Radar based hand gesture recognition further increases complexity due to multi dimensional inputs, specialized preprocessing, and real time requirements. Field-programmable gate arrays (FPGAs) are well suited for such edge inference tasks as they enable customizable datapaths optimized for energy efficiency and throughput. Two main FPGA design paradigms are used for deep learning acceleration. RTL (register transfer level) or SystemVerilog design provides fine-grained control over memory mapping and pipelining to minimize area and power consumption [10]. In contrast, High Level Synthesis (HLS) accelerates development by mapping quantized networks onto pre optimized Deep Learning Processing Unit (DPU) overlays and automating scheduling and buffering [11]. While RTL favors efficiency at the cost of longer development cycles, HLS enables faster deployment and higher operating frequencies with increased LUT and DSP usage. This work deploys a depthwise separable CNN for radar gesture recognition on a Xilinx FPGA using both handcrafted SystemVerilog and HLS based implementations.

The main contributions of the paper are:

1) A compact depthwise separable network architecture optimized for radar gesture inputs and quantization friendly training.
2) A fully modular SystemVerilog implementation of the network, including memory management and a layer wise FSM (Finite State Machine) control scheme.
3) An HLS based deployment, together with an analysis of quantization effects and runtime behavior on the DPU overlay.

II. Related Work

A substantial body of work compares manual HDL (Hardware Description Language) design and HLS driven flows for image processing and CNN accelerators on FPGAs. Prior studies show that HLS can significantly reduce development time and, with appropriate pragmas and expertise, approach HDL quality results, although resource and power trade-offs remain [6]–[8].

Xilinx's Vitis AI and DPU based approach further enable rapid deployment of quantized CNNs on UltraScale and Zynq platforms by converting trained models into DPU instruction streams executed on pre-optimized inference engines, with several works reporting improved throughput and reduced design effort [2], [3]. Complementary studies on HLS based convolution accelerators highlight loop tiling, pipelining, and unrolling as key techniques for maximizing parallelism and memory locality on SoC-FPGA platforms [4]–[6]. Comparative evaluations consistently report that (1) highly optimized HDL designs achieve lower logic usage and power, (2) HLS enables faster development and design space exploration, and (3) vendor DPU toolchains provide fast time to deployment with competitive post quantization accuracy at the cost of higher LUT and DSP usage [7]–[9]. The results presented in this paper align with these trends and provide quantitative evidence for a radar-based gesture recognition workload.

III. Network Architecture

The architecture of the proposed *Radar Hand Gesture Recognition Neural Network* is shown in Fig. 1. The model is optimized for radar-based gesture recognition while maintaining compactness and hardware efficiency for FPGA deployment. Depthwise-separable and grouped convolutions are employed to reduce computational complexity and parameter count without compromising accuracy. Feature extraction is performed using three identical stages, each comprising depthwise and pointwise convolutions followed by batch normalization (BN), Rectified linear unit (ReLU) activation, and max pooling, enabling progressive abstraction of gesture features.

- **Depthwise Convolution:** Channel-wise spatial filtering using a $K \times K$ kernel, $Y_c(i,j) = \sum_{u=0}^{K-1} \sum_{v=0}^{K-1} X_c(i+u, j+v) W_c(u,v)$.
- **Pointwise Convolution (1×1):** Channel aggregation via 1×1 kernels, $Y_k(i,j) = \sum_{c=1}^{C} X_c(i,j) W_{c,k}$.
- **Batch Normalization:** Activation normalization, $\hat{x} = \frac{x-\mu}{\sqrt{\sigma^2+\epsilon}}$, $y = \gamma \hat{x} + \beta$.
- **ReLU:** Non-linear rectification, $\mathrm{ReLU}(x) = \max(0, x)$.
- **Max Pooling:** Spatial downsampling, $Y(i,j) = \max_{u,v \in \{0,1\}} X(2i+u, 2j+v)$.
- **Grouped Convolution:** Group-wise convolution, $Y_g = X_g * W_g$.
- **Fully Connected:** Dense mapping, $y = Wx + b$.
- **Softmax:** Class probability estimation, $p_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$.

TABLE I
DESCRIPTION OF VARIABLES USED IN NETWORK EQUATIONS

Symbol	Meaning
X, Y	Input and output feature maps
$X_c(i,j)$	Input feature value at location (i,j)
$Y_c(i,j)$	Depthwise convolution output at location (i,j)
$Y_k(i,j)$	Pointwise convolution output at location (i,j)
i, j	Spatial indices
u, v	Kernel indices
c, k	Channel indices
K	Convolution kernel size
C	Number of input channels
W, b	Weights and bias
μ, σ^2	Batch normalization mean and variance
γ, β	Batch normalization scale and shift
G	Number of convolution groups
z_i, p_i	Logit and softmax probability for class i

These operations collectively define the computational pipeline used for gesture feature extraction. The evolution of tensor dimensions across all layers is summarized in Table II.

TABLE II
LAYER-BY-LAYER TENSOR DIMENSIONS OF THE PROPOSED NETWORK

Layer	Input Shape	Output Shape
Depthwise Conv 1	$1 \times 192 \times 28 \times 32$	$1 \times 192 \times 28 \times 32$
Pointwise Conv 1	$1 \times 192 \times 28 \times 32$	$1 \times 32 \times 28 \times 32$
BN + ReLU + Pool 1	$1 \times 32 \times 28 \times 32$	$1 \times 32 \times 14 \times 16$
Depthwise Conv 2	$1 \times 32 \times 14 \times 16$	$1 \times 32 \times 14 \times 16$
Pointwise Conv 2	$1 \times 32 \times 14 \times 16$	$1 \times 64 \times 14 \times 16$
BN + ReLU + Pool 2	$1 \times 64 \times 14 \times 16$	$1 \times 64 \times 7 \times 8$
Depthwise Conv 3	$1 \times 64 \times 7 \times 8$	$1 \times 64 \times 7 \times 8$
Pointwise Conv 3	$1 \times 64 \times 7 \times 8$	$1 \times 128 \times 7 \times 8$
BN + ReLU + Pool 3	$1 \times 128 \times 7 \times 8$	$1 \times 128 \times 3 \times 4$
Grouped Conv	$1 \times 128 \times 3 \times 4$	$1 \times 128 \times 3 \times 4$
Flatten	$1 \times 128 \times 3 \times 4$	1536
FC1	1536	128
FC2 (Softmax)	128	5

Overall, the network transforms low level spatial features into high level gesture representations using depthwise separable convolutions and grouped convolution for efficient interchannel interaction. Batch normalization, ReLU activation, and max pooling improve stability and reduce computational cost, while the final fully connected layers perform classification. Dropout is applied during training to improve generalization.

IV. HARDWARE IMPLEMENTATION OF NETWORK ACCELERATOR

This section presents the hardware implementation of the proposed Hand Gesture Recognition Neural Network on the Xilinx ZCU104 FPGA using both HDL and HLS design methodologies. Figures 1, 2, and 3 illustrate the network architecture and corresponding hardware realizations. Both approaches implement the same trained model, enabling a direct comparison of performance, resource utilization, and power efficiency.

A. *HDL based Implementation*

Figure 2 illustrates the handcrafted HDL architecture and centralized finite state machine (FSM) that controls all neural network layers. The accelerator is implemented entirely in SystemVerilog on the Xilinx ZCU104 platform using a modular, layer wise design in which depthwise convolution, pointwise convolution, batch normalization, ReLU with max pooling, grouped convolution, and fully connected layers are realized as independent hardware modules with dedicated control logic and BRAM interfaces. The design follows a hierarchical bottom up methodology, where each layer is derived from its mathematical formulation and mapped to a dedicated module, with intermediate feature maps, weights, and biases stored in on-chip dual-port BRAMs to enable data reuse and minimize external memory access. The first three stages implement depthwise separable convolution with batch normalization, ReLU activation, and 2×2 max pooling, followed by a grouped convolution stage to reduce computational complexity and DSP utilization while preserving inter-channel feature diversity. Two fully connected layers perform final classification, and a centralized FSM enforces deterministic, sequential execution by activating each layer only after the preceding layer asserts its `done` signal. As a result, the accelerator operates as a time-multiplexed hardware pipeline with predictable timing, reduced memory bandwidth, and low power consumption.

B. *HLS based Implementation*

Figure 3 illustrates the HLS-generated hardware architecture comprising the DPU core, memory interfaces, and PS–PL interconnect managed by the Vitis AI toolchain. The same trained neural network is deployed using a High Level Synthesis (HLS) flow on the Zynq UltraScale+ ZCU104 platform, leveraging the pre-optimized DPU to eliminate manual RTL design. Using the Vitis AI toolchain, the model is quantized from 32-bit floating-point to 8-bit fixed-point precision and compiled into optimized instruction streams for the DPUCZDX8G B4096 core, reducing computational complexity and memory bandwidth with minimal accuracy loss. The compiled design is deployed through the Vitis AI runtime, which manages memory allocation, instruction scheduling, and

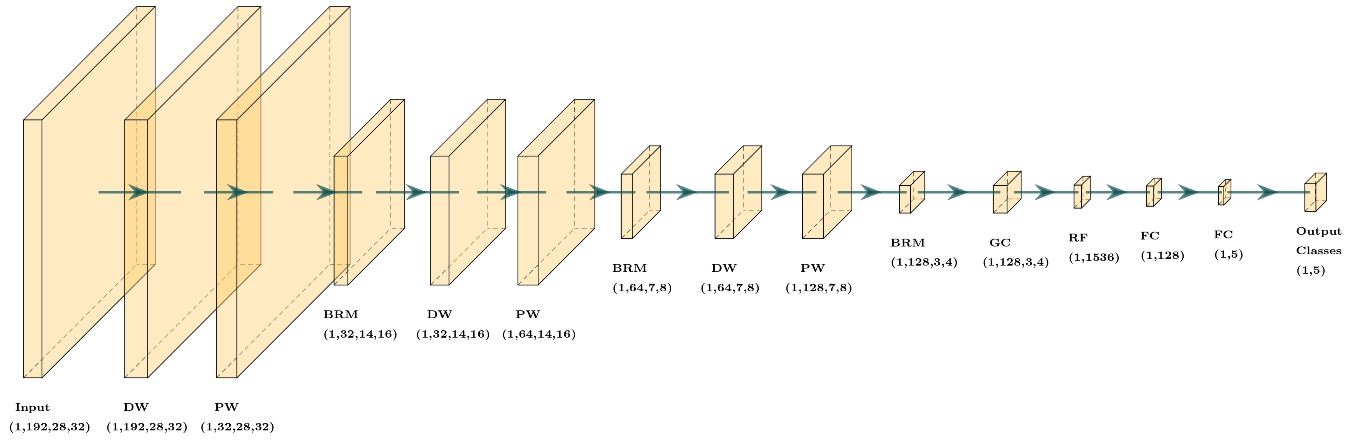

Fig. 1. Block diagram of the proposed *Gesture Recognition Neural Network* architecture. Abbreviations used in the figure: DW - Depthwise Convolution, PW - Pointwise Convolution, BRM - Batch Normalization + ReLU + Max Pooling, GC - Group Convolution, RF - ReLU + Flatten, FC - Fully Connected.

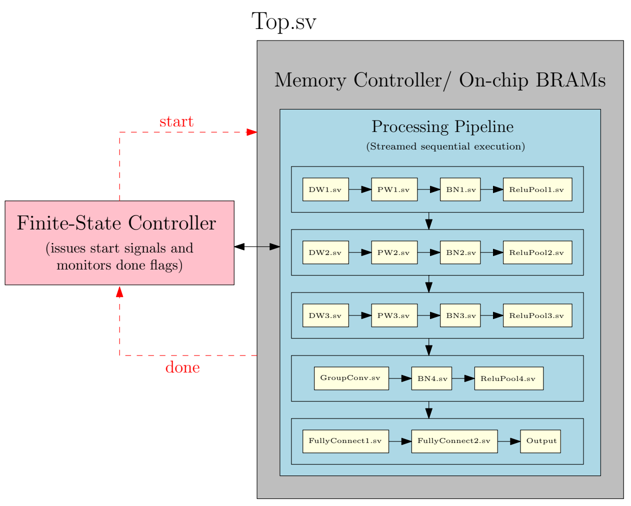

Fig. 2. HDL-generated hardware blocks for the *Gesture Recognition Neural Network* on the ZCU104 platform.

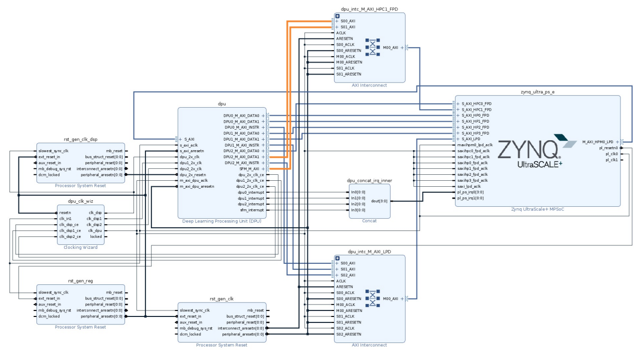

Fig. 3. HLS-generated hardware blocks for the *Gesture Recognition Neural Network* on the ZCU104 platform using the Vitis AI toolchain.

PS–PL communication. The DPU integrates dedicated MAC arrays and functional units for convolution, activation, and pooling, while intermediate feature maps are buffered in on-chip BRAM and URAM. The ARM Cortex-A53 processor acts as the host controller, enabling efficient hardware–software co-design and rapid deployment across FPGA platforms.

V. Results and Discussion

A. Hardware and Software Setup

The proposed neural network was implemented and evaluated on the Xilinx ZCU104 platform featuring a Zynq UltraScale+ MPSoC with a quad-core ARM Cortex-A53 at 1.2 GHz, a dual-core Cortex-R5, and 16 nm FinFET programmable logic, along with 2 GB DDR4 memory and on-chip BRAM and DSP resources. The HDL implementation was synthesized in SystemVerilog using Vivado 2023.1, while the HLS implementation employed Vitis AI 3.0 to compile the trained PyTorch model, quantized using the Vitis AI Quantizer, into executable instructions for the DPUCZDX8G-B4096 core. Both designs were evaluated on the ZCU104 at 95 MHz (HDL) and 300 MHz (HLS), respectively.

TABLE III
Hardware and Software Specifications

Parameter	Specification
FPGA Board	Xilinx ZCU104 (Zynq UltraScale+ MPSoC)
Processor	Quad-core ARM Cortex-A53 @ 1.2 GHz
Programmable Logic	UltraScale+ 16 nm FinFET Fabric
Memory	2 GB DDR4
Software Tools	Vivado 2023.1, Vitis AI 3.0, PyTorch 2.0

B. Dataset Specification

The proposed model was trained and evaluated using the 60 GHz FMCW Radar Gesture Dataset [1], which contains radar reflections from ten dynamic hand gesture classes recorded under varying distances and viewing angles. Raw radar signals were preprocessed into range–Doppler and range–angle representations to generate structured input tensors for the neural network. The dataset comprises approximately 5,000 labeled samples, with nearly 500 instances per class, captured at a sampling rate of 3.072 MSps and a carrier frequency of 60 GHz. The preprocessed data were organized into tensors of size (1, 192, 28, 32), where 192 denotes temporal frames and 28×32 represents spatial resolution. Standard preprocessing steps, including normalization, windowing, and zero-padding, were applied, and an 80%–20% train–test split was used for evaluation.

TABLE IV
DATASET SPECIFICATIONS FOR 60 GHZ FMCW RADAR GESTURE RECOGNITION

Parameter	Value
Dataset Name	60 GHz FMCW Radar Gesture Dataset
Gesture Classes	10
Samples per Class	∼500
Total Samples	5,000
Data Type	Range–Doppler and Range–Angle maps
Input Dimension	(1, 192, 28, 32)
Split Ratio	80% Training / 20% Testing
Sampling Rate	3.072 MSps
Carrier Frequency	60 GHz

C. Comparison with Existing Works (HLS based CNNs)

Table VI compares the proposed HLS based implementation with existing FPGA-based CNN accelerators developed using High-Level Synthesis methodologies. While earlier works primarily focused on image classification, the proposed model extends the approach to radar-based gesture recognition with similar efficiency and accuracy trends.

TABLE V
RESOURCE UTILIZATION AND PERFORMANCE COMPARISON: HDL VS. HLS IMPLEMENTATION

Parameter	HDL Implementation	HLS Implementation
Platform	ZCU104 (UltraScale+)	ZCU104 (UltraScale+)
Frequency (MHz)	95	300
LUTs	10,774	51,843
LUTRAMs	4	–
Flip-Flops (FFs)	5,404	98,567
BRAMs	25	–
URAMs	–	68
DSP Slices	11	710
BUFGs	1	–
Power (W)	0.743	3.623
Accuracy (%)	80.00	81.04

The HLS based flow achieved a higher clock frequency and throughput compared to prior works due to the use of the pre-optimized DPUCZDX8G core. Despite the increase in resource utilization (LUTs and DSPs), the proposed implementation delivered improved accuracy and reduced design time, validating the efficiency of HLS based DPU acceleration on the UltraScale+ platform.

D. Comparison with Existing Works (HDL based CNNs)

Table VII compares the handcrafted HDL implementation with existing low-level CNN accelerators developed using Verilog or VHDL. While traditional HDL designs demonstrate compact resource utilization, they generally exhibit lower operating frequency due to limited pipelining and manual timing closure.

TABLE VI
COMPARISON OF HLS BASED CNN IMPLEMENTATIONS ON FPGA

Work	Platform	Model	Tool	Freq. (MHz)	LUTs	DSPs	Acc. (%)
Ushiroyama et al., 2022 [2]	Zynq-7000 (XC7Z020)	4–84 layer CNN (CIFAR-10)	Vitis AI	250	53k	220	80.2
Tétrault et al., 2018 [8]	ZC702	Sobel+CNN accelerator	Vivado HLS	150	79.6k	140	90.4
Sarg et al., 2021 [4]	Zynq UltraScale+ ZCU104 (ZU7EV)	ResNet-50 Conv Accelerator	Vivado HLS (SD-SoC)	150	120,531	1,331	–
Duarte et al., 2025 [3]	Zynq-7020	7-layer fixed-point CNN	Vivado HLS	200	23k	106	78.5
R. Millón et al., 2020 [7]	Zynq-7000 (ZYBO)	Sobel edge-detection SoC	Vivado HLS	300	51k	160	79.2
Proposed Work	ZCU104 (Ultra-Scale+)	Depthwise-separable CNN	Vitis AI	**300**	**51.8k**	**710**	**81.0**

TABLE VII
COMPARISON OF HDL BASED CNN AND VISION ACCELERATORS ON FPGA

Work	Platform	Model / Task	Freq. (MHz)	LUTs	Power (W)	Acc. (%)
Tsiktsiris et al., 2019 [6]	Cyclone IV / Zynq-7000	CNN for Image Segmentation (VHDL)	85	11.6k	1.5	79.3
Srilakshmi et al., 2023 [9]	Artix-7 (XC7A35T)	4-Layer CNN (Verilog Impl.)	100	144	0.65	78.5
Proposed Work	ZCU104 (Ultra-Scale+)	Depthwise-separable CNN	**95**	**10.8k**	**0.743**	**80.0**

The proposed HDL design achieved competitive accuracy and low power consumption compared to existing handcrafted CNN accelerators. Although it operates at a lower frequency than its HLS counterpart, it offers fine-grained control, reduced logic utilization, and predictable timing behavior. Together, the HDL and HLS implementations demonstrate the trade-offs between manual optimization and automated synthesis, highlighting the evolving balance between resource efficiency, scalability, and development productivity in FPGA-based deep learning systems.

VI. CONCLUSION

The handcrafted HDL implementation achieved timing closure on the ZCU104 with efficient resource usage and predictable performance. This approach offers fine-grained control over the hardware architecture, enabling optimized resource allocation and deterministic timing behavior. In contrast, the HLS based deployment simplifies design development by using a fixed and highly parallel compute structure, allowing faster implementation at the cost of higher resource usage. Overall, the results show a clear trade-off between resource efficiency and performance, helping guide the choice between RTL design and HLS based FPGA deployment based on application requirements and design constraints.

References

[1] K. Apuroop, D. Sahil, S. Murali, S. Ram, S. Muralidhar, and S. Thangavelu, "60 GHz FMCW Radar Gesture Dataset," *IEEE Dataport*, 2020. [Online]. Available: https://ieee-dataport.org/documents/60-ghz-fmcw-radar-gesture-dataset. doi: 10.21227/kc3f-hf85.

[2] A. Ushiroyama, M. Watanabe, N. Watanabe, and A. Nagoya, "Convolutional Neural Network Implementations Using Vitis AI," in *Proc. IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, Las Vegas, NV, USA, 2022, pp. 365–370. doi: 10.1109/CCWC54503.2022.9720794.

[3] D. Duarte, T. Gonçalves, G. Jacinto, P. Flores, and M. Véstias, "Design of Real-Time Gesture Recognition with CNNs on a Low-End FPGA," *Electronics*, vol. 14, no. 17, pp. 1–22, 2025. doi: 10.3390/electronics14173457.

[4] M. Sarg, A. H. Khalil, and H. Mostafa, "Efficient HLS Implementation for Convolutional Neural Networks Accelerator on an SoC," in *Proc. IEEE Int. Conf. on Microelectronics (ICM)*, Giza, Egypt, 2021, pp. 1–4. doi: 10.1109/ICM52667.2021.9664920.

[5] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," Proc. ACM/SIGDA FPGA, 2015.

[6] D. Tsiktsiris, D. Ziouzios, and M. Dasygenis, "A High-Level Synthesis Implementation and Evaluation of an Image Processing Accelerator," *Technologies*, vol. 7, no. 1, p. 4, 2019. doi: 10.3390/technologies7010004.

[7] R. Millón, E. Frati, and E. Rucci, "A Comparative Study Between HLS and HDL on SoC for Image Processing Applications," *Revista Elektron*, vol. 4, no. 2, pp. 100–106, 2020. ISSN: 2525-0159.

[8] M.-A. Tétrault, "Two FPGA Case Studies Comparing High-Level Synthesis and Manual HDL for HEP Applications," in *Proc. IEEE Int. Conf. ReConFigurable Computing and FPGAs (ReConFig)*, Cancún, Mexico, 2018, pp. 1–6. doi: 10.1109/ReConFig.2018.8641624.

[9] S. Srilakshmi, K. S. Prasad, and V. S. R. Krishna, "A Comparative Analysis of HDL and HLS for Developing CNN Accelerators," in *Proc. IEEE Int. Conf. Computational Intelligence and Computing Research (ICCIC)*, Coimbatore, India, 2023, pp. 230–235. doi: 10.1109/ICCIC56738.2023.10024579.

[10] R. Choudhury, S. R. Ahamed, and P. Guha, "Training Accelerator for Two Means Decision Tree," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 7, pp. 1465–1469, 2021. doi: 10.1109/TVLSI.2021.3076081.

[11] R. Choudhury, S. R. Ahamed, and P. Guha, "FPGA Implementation of Low Complexity Hybrid Decision Tree Training Accelerator," in *Proc. IEEE Int. Midwest Symp. Circuits Syst. (MWSCAS)*, 2021, pp. 511–514. doi: 10.1109/MWSCAS47672.2021.9531848.

[12] S. Gupta, R. Poddar, and J. Cong, "High-Level Synthesis for FPGAs: From Prototyping to Deployment," *IEEE Micro*, vol. 40, no. 4, pp. 20–31, Jul.–Aug. 2020. doi: 10.1109/MM.2020.2997606.

[13] J. Cong and B. Xiao, "Minimizing Computation in Convolutional Neural Networks," in *Proc. IEEE Int. Conf. Field-Programmable Technology (FPT)*, 2014, pp. 1–8. doi: 10.1109/FPT.2014.7082778.

[14] A. Canis, J. Choi, M. Aldham et al., "LegUp: High-Level Synthesis for FPGA-Based Processor/Accelerator Systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 6, pp. 1–14, Jun. 2013. doi: 10.1109/TCAD.2013.2244631.

[15] Y. Ma, N. Suda, Y. Cao et al., "An Automatic RTL Compiler for High-Throughput FPGA Implementation of CNNs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 11, pp. 2131–2144, Nov. 2019. doi: 10.1109/TCAD.2018.2880313.

[16] S. I. Venieris and C.-S. Bouganis, "f-CNN: An FPGA-Based Framework for Training Convolutional Neural Networks," in *Proc. IEEE Int. Symp. Field-Programmable Custom Computing Machines (FCCM)*, 2016, pp. 1–8. doi: 10.1109/FCCM.2016.24.

[17] N. J. Fraser, G. Gambardella, and G. Di Guglielmo, "Optimizing CNN Inference on FPGAs," in *Proc. IEEE Int. Symp. Field-Programmable Custom Computing Machines (FCCM)*, 2018, pp. 1–9. doi: 10.1109/FCCM.2018.00014.