

FPGA-Based Implementation of Convolutional Autoencoder

Vemulapalli Sri Samadarsini
jv29859@umbc.edu

University of Maryland Baltimore County
Baltimore, Maryland, USA

Dhandeep Challagundla
vd58139@umbc.edu

University of Maryland Baltimore County
Baltimore, Maryland, USA

Riadul Islam
riaduli@umbc.edu

University of Maryland Baltimore County
Baltimore, Maryland, USA

Abstract—This paper presents the hardware-software co-design and implementation of a convolutional autoencoder on an FPGA platform, aimed at real-time image compression and reconstruction. The architecture consists of 22 layers, including convolution, pooling, transposed convolution, and upsampling stages. A custom processing element (PE) is developed in Verilog to perform 3×3 MAC operations, integrated with a memory-mapped interface for communication with the embedded processor. The system is deployed on the Intel DE1-SoC FPGA, leveraging parallelism and reconfigurability for enhanced performance and power efficiency. The encoder and decoder pipelines are managed through embedded C software. The original float32 Keras model is quantized to an 8-bit integer format using full integer quantization, and these integer parameters are mapped directly to the hardware. The proposed solution effectively preserves reconstruction quality, achieving an average pixel-level accuracy of 87.58%, while delivering significant hardware acceleration with minimal resource utilization.

Index Terms—Convolutional Autoencoder, FPGA Acceleration, Image Compression, Hardware-Software Co-Design, Verilog HDL, DE1-SoC, Embedded Systems, Memory-Mapped I/O.

I. INTRODUCTION

The growing demand for real-time processing in edge applications such as image compression, denoising, and anomaly detection has significantly increased interest in deploying deep learning models on energy-efficient, reconfigurable hardware platforms. Deep neural networks, particularly Convolutional Autoencoders (CAEs), are well-suited for these tasks due to their ability to learn compact feature representations that preserve essential spatial and structural details of input data [1], [2]. This makes CAEs ideal for use in bandwidth- and power-constrained environments like mobile and embedded systems.

In parallel with the development of advanced neural network models, hardware acceleration has emerged as a vital enabler for practical deployment. Field-Programmable Gate Arrays (FPGAs) provide a compelling solution for implementing deep learning algorithms, offering inherent parallelism, customizable datapaths, and lower power consumption compared to traditional CPU or GPU architectures [3], [4]. FPGAs have demonstrated superior energy efficiency for workloads involving repetitive computations, such as convolution operations, making them particularly appealing for implementing CNNs and CAEs in real-time scenarios [5].

However, deploying high-depth convolutional architectures on FPGAs introduces several challenges, including efficient memory access, latency control, resource-aware quantization,

and scalable hardware design. Non-uniform quantization techniques [5] and automated hardware-aware optimizations [6] have been explored to address these limitations, highlighting the need for integrated design methodologies that co-optimize model architecture and hardware implementation.

This work presents a hardware-software co-designed implementation of a 22-layer convolutional autoencoder on the Intel DE1-SoC FPGA platform. The architecture includes convolution, max pooling, upsampling, and transposed convolution layers. A custom 3×3 Processing Element (PE), developed in Verilog, is used to accelerate multiply-accumulate (MAC) operations. Communication between the FPGA fabric and the ARM Cortex-A9 processor is managed via memory-mapped I/O using the Avalon-MM interface. All intermediate data handling and control logic, including activation functions and sampling layers, are managed in embedded C software.

The original Keras-based float32 model is quantized using full integer quantization to convert all weights, inputs, and outputs to int8 format, with biases in int32, ensuring compatibility with hardware constraints and enhancing execution efficiency [7]. These quantized parameters are statically mapped to the hardware, eliminating runtime software dependency. The resulting system achieves hardware acceleration while preserving reconstruction accuracy, making it suitable for intelligent vision applications in real-time, edge-based deployments [2], [4].

A. Motivation

Convolutional autoencoders (CAEs) are widely used in image processing tasks such as compression, denoising, and anomaly detection, due to their ability to extract compact and meaningful feature representations [1]. However, deploying such deep architectures in real-time edge environments introduces critical challenges related to latency, memory, and energy efficiency.

Field-Programmable Gate Arrays (FPGAs) have emerged as a compelling alternative to CPUs and GPUs for such workloads, offering advantages such as hardware-level parallelism, customizable datapaths, and low power consumption [3], [4]. Previous works have shown that FPGA-based implementations of neural networks can significantly reduce inference latency and energy use, while maintaining accuracy.

Recent studies like NLIC [5] and Balle's nonlinear transforms [2] have explored lightweight and learned compression methods, emphasizing the importance of hardware-aware

quantization and efficient transforms. Additionally, real-time FPGA accelerators for CNNs and compression frameworks [6] underline the potential of fully offloading compute-intensive operations to hardware for high-throughput applications.

Motivated by these advancements, this work aims to accelerate a 22-layer convolutional autoencoder entirely in hardware using an Intel DE1-SoC FPGA. The model is trained in Keras using 32-bit floating-point precision and subsequently quantized to an 8-bit integer format. The system uses only quantized weights and biases from the trained model, while all core operations convolution, pooling, upsampling, and transposed convolution—are offloaded to custom Verilog based hardware modules. This approach is intended to achieve both computational efficiency and scalability for edge deployment.

B. Contributions

The main contributions of this work are as follows:

- Design and hardware acceleration of a 22-layer convolutional autoencoder for real-time image compression and reconstruction, optimized for FPGA execution.
- Development of a custom Verilog-based processing element (PE) for 3×3 multiply-accumulate (MAC) operations, integrated through a memory-mapped Avalon-MM interface.
- Hardware-software co-design methodology implemented on the Intel DE1-SoC platform, with compute-intensive layers executed on FPGA fabric and control/memory management handled by the Hard Processor System (HPS) using embedded C.
- Parameter extraction from a fully quantized Keras model, with INT8 precision for inputs, weights, and outputs, and INT32 for biases, and integrated these values into the hardware model.
- Complete end-to-end hardware implementation of the encoder and decoder pipelines, covering convolution, ReLU, pooling, transposed convolution, and upsampling operations.
- Comprehensive analysis of performance metrics including execution time, FPGA resource utilization, and reconstruction accuracy over multiple image inputs.

II. BACKGROUND

Convolutional autoencoders (CAEs) have emerged as powerful tools in unsupervised learning for tasks such as image compression, denoising, and anomaly detection [1]. These networks consist of an encoder that reduces the spatial dimensionality of input data, followed by a decoder that reconstructs the original data from a compact latent representation. Unlike fully connected autoencoders, CAEs exploit spatial hierarchies using convolutional and transposed convolutional layers, making them highly effective for visual data processing [8].

While CAEs are traditionally implemented in software using GPUs or CPUs, such platforms often struggle to meet the stringent power and latency requirements of edge applications. This limitation has driven interest in implementing CAEs on Field-Programmable Gate Arrays (FPGAs), which offer

substantial benefits in terms of energy efficiency, deterministic latency, and hardware-level parallelism [9], [10].

A range of prior works has demonstrated the feasibility and advantages of FPGA-based CAEs. For instance, Isik et al. [11] implemented an energy-efficient reconfigurable CAE using VHDL to denoise MNIST images on an FPGA, achieving a throughput of 21.12 GOP/s while consuming only 5.93 W of power. Their design used fixed-point arithmetic and pipeline parallelism to achieve high efficiency.

Huang et al. [12] proposed a layer-multiplexing strategy that enables convolutional and deconvolutional layers to share the same hardware blocks. This significantly reduces the design footprint and improves scalability, which is crucial for high-layer-count architectures.

Fan et al. [13] further explored hybrid quantization and knowledge distillation techniques to compress deep models for FPGA inference. Their results indicated that low-bit quantization and architectural simplification enable real-time performance with minimal accuracy degradation.

Han et al. [14] expanded on the design space with an energy-efficient FPGA-based autoencoder optimized for on-chip training and inference. Their solution adopted INT8 quantization and fixed-point training, reducing memory bandwidth while maintaining high reconstruction quality through resource-multiplexed convolution modules.

Recent trends have also moved toward secure and reconfigurable autoencoders, such as the work by Mazouz et al. [6], who demonstrated FPGA-integrated learned image compression solutions with support for secure transmission and high-throughput inference.

In addition, non-uniform quantization techniques have been explored by Ge et al. [5], who developed a learned compression method using optimized quantizers and lightweight transforms suitable for hardware deployment. Similarly, Balle [2] investigated efficient nonlinear transforms for image compression that reduce redundancy while preserving perceptual quality.

Broader comparisons of hardware platforms have also been reported. Fowers et al. [3] examined the performance and energy characteristics of FPGAs versus GPUs and multicore CPUs in convolutional workloads, highlighting the superior energy efficiency of FPGAs for sliding-window-based models.

These studies underscore the importance of hardware-software co-design for deep learning on edge platforms. The present work builds on these foundations by implementing a 22-layer convolutional autoencoder on the Intel DE1-SoC FPGA. The system incorporates custom Verilog modules for 3×3 MAC operations and C-based control on the Hard Processor System (HPS). Unlike hybrid solutions that rely on runtime inference engines, this approach fully maps the quantized model into hardware using INT8 parameters and INT32 biases, enabling real-time, deterministic execution with minimal runtime overhead.

III. AUTOENCODER ARCHITECTURE

A. Autoencoder

Autoencoders are a type of unsupervised neural network used for various representation learning tasks such as image

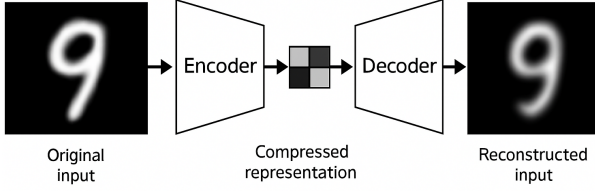


Fig. 1: Autoencoder architecture depicting the compression and reconstruction path, adapted from [14].

compression, denoising, and anomaly detection [1]. They consist of two main parts: an encoder that compresses the input into a latent space representation, and a decoder that reconstructs the original data from this representation. The typical autoencoder structure includes three key phases:

- Encode and Compress
- Bottleneck Representation
- Decode and Reconstruct

The encoder typically reduces the spatial resolution of the input using transformations such as convolutions and pooling. The resulting bottleneck captures the essential features in a compact form. The decoder, in turn, applies transformations like upsampling and transposed convolutions to reconstruct the original input. This architecture enables lossy compression while preserving the most significant features of the data.

1) *Simple Autoencoder*: Simple autoencoders are built using fully connected (dense) layers and represent the foundational form of autoencoders. These networks consist of an input layer, one or more hidden layers for encoding, a bottleneck layer, and symmetrical layers for decoding. Although effective for low-dimensional data, fully connected autoencoders do not capture spatial hierarchies well, making them less suitable for image data.

2) *Convolutional Autoencoder*: Convolutional autoencoders (CAEs) extend the functionality of simple autoencoders by replacing the fully connected layers with convolutional operations, making them particularly effective for spatial data such as images [14]. A typical CAE consists of an encoder that performs feature extraction and spatial downsampling, and a decoder that reconstructs the input image by reversing these transformations.

The encoder section consists of layers such as:

- Convolutional layers (for spatial feature extraction)
- Rectified Linear Unit (ReLU) activation
- MaxPooling (for spatial downsampling)

The decoder typically reverses these operations using:

- Transposed convolutional layers (for upsampling)
- ReLU activation
- Output layer with sigmoid or linear activation depending on the task

The building blocks of a convolutional autoencoder include the following operations:

1. Convolution: This layer extracts features by sliding a kernel over the input and computing dot products. The 2D convolution operation is mathematically defined as:

$$Y(i, j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X(i+m, j+n) \cdot K(m, n) \quad (1)$$

where X is the input feature map, K is the convolution kernel, and $Y(i, j)$ is the output feature at position (i, j) .

2. ReLU (Rectified Linear Unit): The ReLU activation introduces non-linearity by zeroing out negative values:

$$\text{ReLU}(x) = \max(0, x) \quad (2)$$

3. Max Pooling: This operation reduces the spatial dimensions by selecting the maximum value within a window of size $k \times k$:

$$Y(i, j) = \max_{\substack{0 \leq m < k \\ 0 \leq n < k}} X(i \cdot s + m, j \cdot s + n) \quad (3)$$

where s is the stride, and X is the input feature map.

4. Transposed Convolution: Also known as deconvolution, this operation is used in the decoder to upsample the feature maps. It is essentially the gradient of a regular convolution with respect to its input and is expressed as:

$$Y(i, j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X(i-m, j-n) \cdot K(m, n) \quad (4)$$

5. UpSampling: This operation increases the spatial resolution of the feature map by repeating values. For nearest-neighbor upsampling:

$$Y(i \cdot r, j \cdot r) = X(i, j) \quad (5)$$

where r is the upsampling factor.

6. Sigmoid Activation: Used in the final output layer to constrain pixel values between 0 and 1:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (6)$$

These components work in tandem to transform the input into a compact latent representation and then reconstruct it. Pooling operations reduce spatial size and increase receptive field, while transposed convolutions help restore the spatial resolution. This architectural design is well-suited for image reconstruction tasks, especially in resource-constrained edge devices. In this work, I utilize all of the above operations to construct our 22-layer CAE architecture optimized for FPGA implementation.

B. System Architecture

The proposed autoencoder architecture consists of 22 layers, including both encoder and decoder stages, optimized for image compression and reconstruction. Fig. 2 shows the design follows a symmetric structure with 5 encoding blocks, a latent space representation, and 5 decoding blocks, each comprising convolutional or transposed convolutional layers, followed by activation and sampling operations.

The encoder receives a grayscale image input of size $416 \times 416 \times 1$ and reduces its spatial resolution using a series

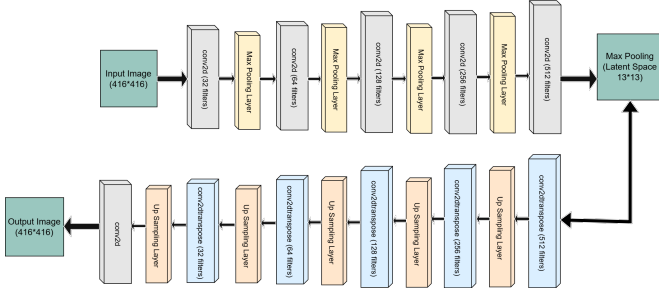


Fig. 2: Overall architecture of the 22-layer convolutional autoencoder

of convolutional layers with increasing filter depths—32, 64, 128, 256, and 512—each followed by a ReLU activation and MaxPooling2D. The output of the encoder is a compressed latent space of dimension $13 \times 13 \times 512$. The decoder mirrors this structure using Conv2DTranspose layers with decreasing filter sizes—512, 256, 128, 64, and 32—followed by ReLU activations and UpSampling2D layers. A final Conv2D layer with one filter and sigmoid activation reconstructs the output, restoring the original input shape. This configuration achieves efficient image compression while preserving reconstruction quality, enabling the design to be highly effective for hardware acceleration on FPGAs.

TABLE I: Layer-wise dimensions of the 22-layer autoencoder

Layer Name	Shape
Input	(416, 416, 1)
Conv2D (32 filters)	(416, 416, 32)
MaxPooling2D	(208, 208, 32)
Conv2D (64 filters)	(208, 208, 64)
MaxPooling2D	(104, 104, 64)
Conv2D (128 filters)	(104, 104, 128)
MaxPooling2D	(52, 52, 128)
Conv2D (256 filters)	(52, 52, 256)
MaxPooling2D	(26, 26, 256)
Conv2D (512 filters)	(26, 26, 512)
MaxPooling2D (Latent space)	(13, 13, 512)
Conv2DTranspose (512 filters)	(13, 13, 512)
UpSampling2D	(26, 26, 512)
Conv2DTranspose (256 filters)	(26, 26, 256)
UpSampling2D	(52, 52, 256)
Conv2DTranspose (128 filters)	(52, 52, 128)
UpSampling2D	(104, 104, 128)
Conv2DTranspose (64 filters)	(104, 104, 64)
UpSampling2D	(208, 208, 64)
Conv2DTranspose (32 filters)	(208, 208, 32)
UpSampling2D	(416, 416, 32)
Conv2D (Final Output Layer)	(416, 416, 1)

Quantized Parameters: The model was initially trained in Keras using 32-bit floating-point precision and later quantized to an 8-bit integer format using full integer quantization. This quantized representation includes:

- INT8 weights and activations
- INT32 biases

These quantized parameters were extracted and statically embedded into the FPGA design, enabling full hardware acceleration with minimal runtime overhead.

Design Highlights:

- Latent space of shape $13 \times 13 \times 512$ enables high compression ratio
- Symmetric encoder-decoder pipeline supports progressive abstraction and reconstruction
- Efficient reuse of convolutional hardware across layers
- Fixed-point arithmetic with dynamic shift logic ensures numerical stability during MAC operations

This architecture balances reconstruction fidelity with computational efficiency, making it highly suitable for real-time edge deployment on resource-constrained platforms.

IV. PROPOSED METHODOLOGY

This section describes the hardware-software co-design approach used to implement the proposed 22-layer convolutional autoencoder on the Intel DE1-SoC FPGA. The design leverages the FPGA fabric for accelerating compute-intensive MAC operations, while control logic and auxiliary computations are handled by the ARM Cortex-A9 processor.

A. Layer Decomposition and Execution Strategy

Each convolutional and transposed convolutional layer in the network is decomposed into a series of 3×3 kernel operations. The ARM processor extracts 3×3 patches from the input feature maps and pairs them with the corresponding quantized filters. These inputs are then transmitted to the FPGA where custom hardware accelerators perform the MAC operations. Once the computation is complete, the result is retrieved and post-processed in software.

Fig. 3 illustrates the system-level dataflow. The ARM HPS handles memory management, patch extraction, post-processing (such as ReLU/sigmoid and pooling), while the FPGA executes MAC operations via dedicated processing elements. This separation ensures computational efficiency and modularity in execution.

B. Processing Element (PE) Role

The FPGA implements multiple Processing Elements (PEs), each designed to perform 3×3 MAC operations in hardware. These PEs accept nine input pixels and their corresponding filter weights, compute parallel multiplications, and accumulate the results using an optimized adder tree.

Fig. 4 shows the internal MAC structure of a PE, where multiplication results are hierarchically summed and registered. This architecture ensures pipelined and low-latency accumulation.

Each PE consumes 5 DSP blocks. As the Intel DE1-SoC FPGA contains 87 DSP blocks, a maximum of 17 PEs can be instantiated in parallel. This degree of concurrency enables the system to compute multiple filters simultaneously, dramatically reducing total inference time.

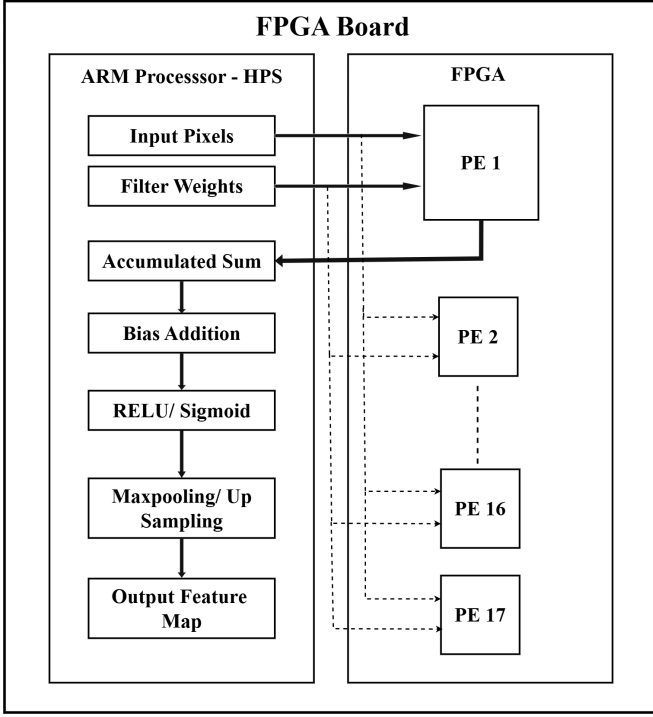


Fig. 3: System-level hardware-software co-design of the FPGA-based autoencoder

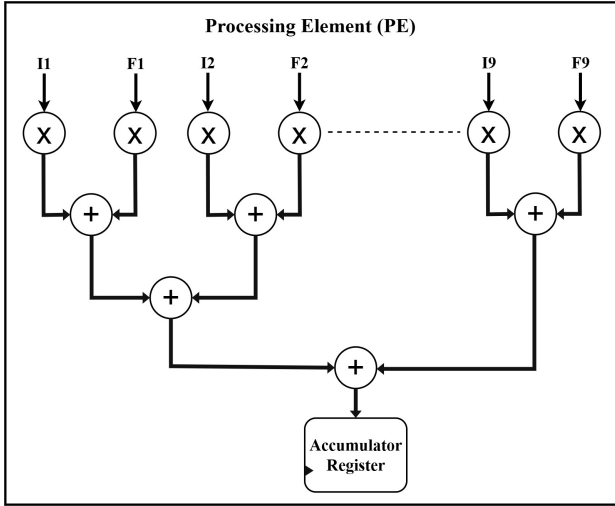


Fig. 4: Adder tree structure within each Processing Element (PE)

C. Software Responsibilities

The embedded C software running on the ARM processor handles tasks such as input patch generation, bias addition, dynamic right-shift scaling, and activation functions. After retrieving the output from the PE, the processor applies post-processing such as ReLU or sigmoid activation depending on the layer. It also performs spatial operations like max pooling and upsampling entirely in software. This allows for easy programmability while minimizing hardware complexity. Additionally, all layer-level execution flow and buffer management are orchestrated by the software layer.

D. Quantized Parameter Integration

The neural network is initially trained in floating-point precision and then quantized to use integer-only arithmetic for FPGA compatibility. Specifically, INT8 is used for weights, inputs, and outputs, while biases are stored as INT32. These quantized values are extracted from a Keras model and embedded directly into the C code, ensuring fully deterministic execution. As no floating-point arithmetic is used, the system maintains computational efficiency while preserving reasonable reconstruction accuracy.

E. Execution Flow Summary

The following steps outline the overall execution sequence for each layer:

- The ARM processor selects a 3×3 input patch and the corresponding filter.
- The patch and filter weights are sent to a PE on the FPGA.
- The PE computes the MAC result and returns it to the processor.
- The processor adds the bias, applies dynamic shift scaling, and performs the ReLU or sigmoid activation.
- Max pooling or upsampling is performed in software if the layer requires it.
- The result is stored and passed as input to the next layer.

This hardware-software co-implementation ensures computationally efficient, low-latency processing while remaining resource-conscious and scalable for future extensions.

V. EXPERIMENTAL SETUP

The proposed hardware-software co-implementation of the 22-layer convolutional autoencoder is deployed and tested on the Intel DE1-SoC FPGA development board, which integrates a Cyclone V FPGA and a dual-core ARM Cortex-A9 Hard Processor System (HPS). Intel Quartus Prime software is used for hardware synthesis, design compilation, and signal integration, while the embedded software is written and compiled using the ARM Linux toolchain.

A. Hardware Platform

The FPGA design, including the custom Processing Elements (PEs), is implemented in Verilog and compiled using Intel Quartus Prime. The DE1-SoC board serves as the target platform, with the HPS executing embedded C code and orchestrating the control flow via memory-mapped interfaces. The board includes USB, HDMI, Ethernet, and GPIO peripherals, and it is used for interfacing with the host machine and debugging during runtime. Fig. 5 shows the DE1-SoC development board.

B. Execution Environment

All layer-wise computations, including patch extraction and output feature collection, are initiated from the HPS and executed using embedded C code. The Linux terminal (via UART serial interface) is used for interacting with the board, executing compiled binaries, and monitoring execution flow.

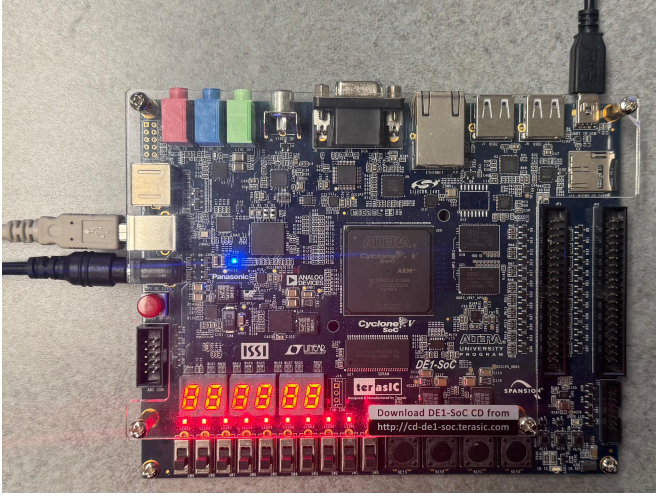


Fig. 5: Intel DE1-SoC FPGA Development Board used for implementation.

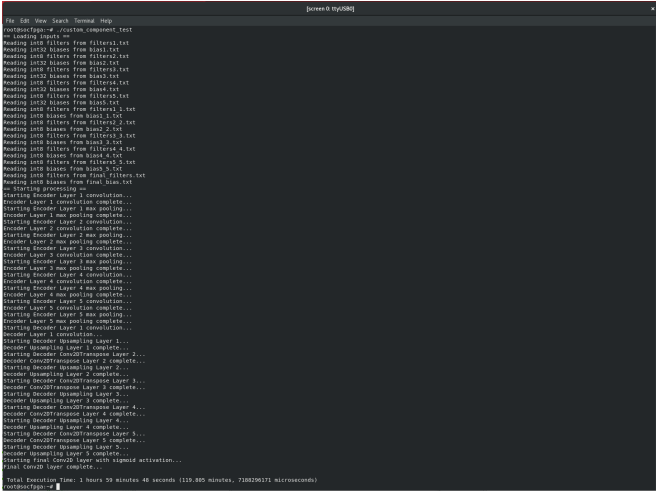


Fig. 6: Execution log of the convolutional autoencoder on the DE1-SoC from UART terminal.

The application runs from the Linux shell environment on the ARM processor, with UART serving as the main communication and debugging channel.

Fig. 6 illustrates a snapshot of the execution log captured from the U-Boot terminal, showing the layer-wise processing updates, including convolution, activation, pooling, and up-sampling stages.

VI. RESULTS AND ANALYSIS

This section presents the hardware resource utilization, qualitative image reconstruction outcomes, and the quantitative evaluation of the proposed convolutional autoencoder (CAE) implemented on the Intel DE1-SoC FPGA.

A. FPGA Resource Utilization

The hardware design was synthesized and implemented using Intel Quartus Prime for the Intel DE1-SoC board based on the Cyclone V FPGA. Table II presents the utilization of various hardware resources after place and route.

TABLE II: FPGA Resource Utilization Summary

Resource	Usage	Percentage
Logic Elements (ALMs)	7,675 / 32,070	24%
M10K Memory Blocks	68 / 397	17%
Total Block Memory Bits	696,320 / 4,065,280	17%
DSP Blocks	85 / 87	98%

Each Processing Element (PE) is implemented using 5 DSP blocks to perform 3×3 MAC operations efficiently. Given the FPGA's availability of 87 DSP blocks, the system can support up to 17 PEs operating in parallel. This configuration enables high-throughput convolution and transposed convolution processing across all layers of the autoencoder.

B. Results

C. Results

To evaluate the performance of the proposed hardware-accelerated convolutional autoencoder, we tested the system on four edge-detected grayscale images. Each image was passed through the 22-layer quantized CAE deployed on the FPGA, and the reconstructed outputs were obtained after decoding.

Visual comparisons between the input and reconstructed images are shown in Table III. The reconstructed outputs demonstrate effective preservation of structural content, despite the use of aggressive quantization and integer-only arithmetic.

To quantify performance, pixel-wise accuracy was calculated as the ratio of matching pixels between the input and the reconstructed output. The accuracy results for each image are summarized in Table IV. The system achieved an average pixel-level accuracy of **87.58%**, indicating reliable reconstruction and validating the effectiveness of the hardware-software co-design.

This level of accuracy confirms that even with low-precision arithmetic (INT8 inputs, weights, outputs and INT32 biases), the proposed architecture can maintain high fidelity in reconstructed images. It highlights the robustness of the quantization strategy and the capability of the FPGA-based PE to handle MAC operations with minimal loss in quality. Furthermore, the consistent results across multiple test images underscore the generalizability of the design, making it suitable for a variety of edge computing scenarios that demand low-latency and power-efficient deep learning inference.




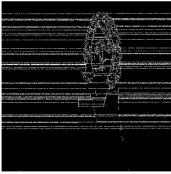
D. Analysis

The results demonstrate that the proposed FPGA-accelerated convolutional autoencoder effectively reconstructs edge-detected grayscale images, achieving an average pixel-level accuracy of 87.58%. This indicates that the structural and spatial information in the input images is largely preserved through the encoding and decoding process.

However, visual inspection of the reconstructed images reveals noticeable horizontal noise patterns. This degradation is more pronounced in background areas or low-intensity regions of the image. Such artifacts can be attributed to several contributing factors:

- *Quantization Effects:* The model parameters and activations are quantized to INT8 to facilitate efficient FPGA

TABLE III: Input vs Output Image Comparison with Accuracy

	Input 1	Input 2
Input Image		
Reconstructed Image		
Accuracy (%)	81.16	90.44

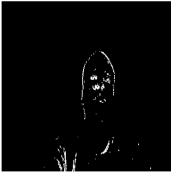

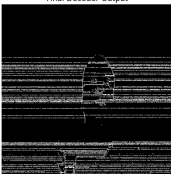
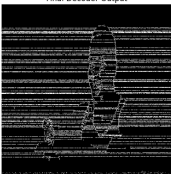
	Input 3	Input 4
Input Image		
Reconstructed Image		
Accuracy (%)	90.14	86.58

TABLE IV: Pixel-Level Reconstruction Accuracy

Image	Total Pixels	Matched Pixels	Accuracy (%)
Input 1	173,056	140,449	81.16
Input 2	173,056	156,515	90.44
Input 3	173,056	155,993	90.14
Input 4	173,056	149,825	86.58
Average	—	—	87.58

execution. This low-precision representation can cause cumulative rounding errors, especially in deeper layers of the network.

- **Integer Arithmetic in PE:** All MAC operations within the Processing Elements are performed using integer arithmetic. This avoids floating-point computation but introduces a loss of precision, which may lead to inaccurate accumulation results and affect feature representation.
- **Fixed-Point Scaling:** Post-processing includes dynamic right-shift scaling to emulate fixed-point behavior. Improper calibration or limited bit-width during scaling can amplify noise in the final output.

Despite these limitations, the decoder is able to reconstruct images that are structurally consistent with their respective inputs. Future work could explore hybrid quantization schemes or selective precision scaling to further reduce noise while preserving efficiency.

VII. CONCLUSION

This paper presented an efficient hardware-software implementation of a 22-layer convolutional autoencoder designed for real-time image compression and reconstruction on the Intel DE1-SoC FPGA platform. By offloading compute-intensive MAC operations to custom Verilog-based Processing Elements and handling control flow and auxiliary operations in embedded C, the design achieved a balanced trade-off between performance and resource efficiency. The use of quantized INT8 parameters enabled compact and fast execution, while maintaining a high average reconstruction accuracy of 87.58% across test cases. Although some visual artifacts were observed, primarily due to fixed-point arithmetic and quantization, the reconstructed outputs successfully preserved key structural features. This work demonstrates the practical potential of deploying deep learning models on reconfigurable hardware. Future efforts can focus on improving visual quality through hybrid precision techniques, advanced noise reduction methods, and extending the design for larger models or more complex image processing tasks.

REFERENCES

- [1] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," *arXiv preprint arXiv:2003.05991*, 2020.
- [2] J. Ballé, "Efficient nonlinear transforms for lossy image compression," in *2018 Picture Coding Symposium (PCS)*. IEEE, 2018, pp. 248–252.
- [3] J. Fowers, G. Brown, P. Cooke, and G. Stitt, "A performance and energy comparison of fpgas, gpus, and multicores for sliding-window applications," in *Proceedings of the ACM/SIGDA international symposium on Field-programmable gate arrays*, 2014, pp. 47–56.
- [4] F. Gan, H. Zuyi, C. Song, and W. Feng, "Energy-efficient and high throughput fpga-based accelerator for convolutional neural networks," in *2016 13th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*. IEEE, 2016, pp. 624–626.
- [5] Z. Ge, S. Ma, W. Gao, J. Pan, and C. Jia, "Nlic: Non-uniform quantization based learned image compression," *IEEE Transactions on Circuits and Systems for Video Technology*, 2024.
- [6] A. Mazouz, C. Tria, S. Chaudhuri, A. Fiandrotti, M. Cagnanzzo, M. Mitrea, and E. Tartaglione, "Security and real-time fpga integration for learned image compression," *arXiv preprint arXiv:2503.04867*, 2025.
- [7] A. Mazouz and C. Bridges, "Automated cnn back-propagation pipeline generation for fpga online training," *Journal of Signal Processing Systems*, vol. 18, pp. 2583–2599, 2024, <https://doi.org/10.1007/>.
- [8] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Better autoencoders for learning compressed representations," *arXiv preprint arXiv:1803.06975*, 2018.
- [9] R. Woods, J. McAllister, G. Lightbody, and Y. Yi, *FPGA-based system design*. John Wiley & Sons, 2008.
- [10] S. M. Trimberger, "Three ages of fpgas: A retrospective on the first thirty years of fpga technology," *Proceedings of the IEEE*, vol. 103, no. 3, pp. 318–331, 2015.
- [11] M. Isik and A. Yazici, "Energy-efficient reconfigurable hardware implementation of convolutional autoencoders for denoising," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2023.
- [12] C. Huang, S. Yin, and S. Wei, "An fpga-based scalable architecture for convolutional autoencoders," in *2018 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2018, pp. 1–4.
- [13] X. Fan, X. Zhou, and M. Xu, "Lightweight neural image compression via hybrid quantization and fpga-aware optimization," *IEEE Transactions on Image Processing*, 2024, early access.
- [14] J. Han and et al., "An energy-efficient reconfigurable autoencoder implementation on fpga," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2023.