International Journal of Computer Engineering in Research Trends
ISSN: 2349-7084/ https://doi.org/10.22362/ijcert/2026/v13/i3/v13i304
Volume 13, Issue 3, March 2026, Pp.29-39

Research Paper

VLSI-Based Parallel CNN Accelerator with Quantization for High-Performance Edge Intelligence

[1*] Jameer Shaik, [2] Anumolu Lasmika, [3] Nalukurthi Sumalatha, [4] Vijaya Lakshmi.C

[*1]*Academic Consultants, Department of Electronics and Communication Engineering, Sri Venkateswara University College of Engineering, Sri Venkateswara University Tirupati, Andhra Pradesh , India*
roshanjameer87@gmail.com
[2]*Academic Consultants, Department of Electronics and Communication Engineering, Sri Venkateswara University College of Engineering, Sri Venkateswara University Tirupati, Andhra Pradesh , India*
anumolu.lasmika@gmail.com
[3]*Academic Consultants, Department of Electronics and Communication Engineering, Sri Venkateswara University College of Engineering, Sri Venkateswara University Tirupati, Andhra Pradesh , India*
nalukurthisumalatha@gmail.com
[4]*Academic Consultants, Department of Electronics and Communication Engineering, Sri Venkateswara University College of Engineering, Sri Venkateswara University Tirupati, Andhra Pradesh , India*
vijayalakshmicvl@gmail.com

**Corresponding Author: roshanjameer87@gmail.com*

Received: 13/12/2025, Revised: 19 /01/2026, Accepted: 22/03/2026 Published: 31/03/2026

Abstract: - The increasing use of deep learning models, especially Convolutional Neural Networks (CNNs), has created a demand for efficient hardware solutions due to their high computational and energy requirements. Traditional CPU and GPU-based systems often face challenges such as high latency and power consumption, particularly in edge devices. The objective of this study is to design and implement an energy-efficient CNN hardware accelerator using VLSI architecture suitable for real-time image classification tasks. The proposed approach integrates a lightweight CNN model with a VLSI-based hardware design that includes parallel processing elements, optimized dataflow, and fixed-point quantization. The system is evaluated using the CIFAR-10 dataset, which consists of 60,000 images across 10 classes. Preprocessing techniques such as normalization and data augmentation are applied, and the trained model is mapped onto hardware using an efficient pipeline. Experimental results show that the proposed system achieves an accuracy of 94.8%, precision of 94.1%, recall of 93.6%, and F1-score of 93.8%. Compared to conventional approaches, the design demonstrates reduced latency and lower power consumption while maintaining high throughput. The use of quantization significantly improves energy efficiency with minimal impact on accuracy. In conclusion, the proposed VLSI-based CNN accelerator provides a practical solution for real-time edge AI applications, offering a balanced trade-off between performance and energy efficiency. This work contributes to the development of scalable and hardware-efficient deep learning systems.

Keywords- CNN Accelerator, VLSI Architecture, Edge AI, Deep Learning, FPGA, Energy Efficiency, Image Classification, Hardware Optimization.

---

1. Introduction

In recent years, the rapid growth of artificial intelligence, especially deep learning, has transformed several application domains such as healthcare, surveillance, autonomous driving, and smart agriculture. Among various deep learning models, Convolutional Neural Networks (CNNs) have become the most widely used technique for image classification and pattern recognition tasks due to their high accuracy and robustness [19]. However, the computational requirements of CNNs are extremely high, involving billions of multiply-accumulate (MAC) operations, which makes their execution challenging on conventional processing platforms.

Traditional processors such as CPUs and GPUs are designed for general-purpose computing and are not optimized for the repetitive and parallel nature of CNN

operations. As a result, these platforms often suffer from high latency, increased energy consumption, and inefficient resource utilization when executing deep learning workloads [4]. This issue becomes more critical in edge devices, where power, memory, and computational resources are limited. Therefore, there is a growing need for specialized hardware solutions that can efficiently handle CNN workloads while maintaining low power consumption and high performance. To address these challenges, researchers have explored VLSI-based hardware accelerators that are specifically designed for deep learning applications. These accelerators leverage parallel processing, optimized dataflow, and efficient memory management techniques to improve computational efficiency. Architectures such as dedicated neural processing units and domain-specific accelerators have demonstrated significant improvements in performance and energy efficiency [5]. However, designing such accelerators requires careful consideration of trade-offs between power, area, and performance.

Despite significant progress in the field of hardware acceleration, several challenges still exist in designing efficient CNN accelerators. One of the major challenges is the high computational complexity associated with convolution operations. Each convolution layer involves a large number of MAC operations, which increases execution time and power consumption. Although parallel processing techniques can reduce computation time, they often lead to increased hardware complexity and area overhead [1]. Another critical challenge is memory access bottleneck. CNN models require frequent access to large volumes of data, including input feature maps, weights, and intermediate outputs. Accessing off-chip memory is energy-intensive and introduces significant latency. Even advanced architectures struggle to minimize memory bandwidth requirements while maintaining high throughput [3]. Efficient data reuse and on-chip buffering techniques are essential to address this issue, but they require sophisticated design strategies.

In addition to computation and memory challenges, power consumption remains a key concern, especially for edge devices. Many existing solutions achieve high performance at the cost of increased energy usage, which is not suitable for battery-operated systems. Techniques such as quantization, sparsity exploitation, and approximate computing have been proposed to reduce power consumption, but they often affect model accuracy if not carefully implemented [18]. Furthermore, scalability and flexibility are important considerations in modern hardware design. CNN models are continuously evolving, with deeper architectures and varying layer configurations. Designing a fixed hardware architecture that can efficiently support multiple CNN models is a challenging task. Existing FPGA-based and ASIC-based solutions provide some level of flexibility, but they often require manual reconfiguration and optimization [2].

Several hardware architectures have been proposed to accelerate CNN computations, including FPGA-based designs, ASIC implementations, and systolic array architectures. While these approaches have shown promising results, they also have certain limitations. For instance, FPGA-based accelerators provide flexibility and reconfigurability, but they may not achieve the same level of performance and energy efficiency as custom ASIC designs [15]. On the other hand, ASIC-based accelerators offer high performance but lack flexibility and require significant design effort. Dataflow-based architectures, such as energy-efficient CNN accelerators, have improved memory utilization by reducing data movement. However, these designs are often optimized for specific CNN models and may not generalize well to other architectures [3]. Similarly, systolic array-based accelerators have demonstrated high throughput by exploiting parallelism, but they require careful scheduling and data management to avoid inefficiencies [16].

Recent works have also explored binarized and sparse neural networks to reduce computational complexity. These approaches significantly reduce the number of operations and memory requirements, but they may lead to degradation in model accuracy if not properly optimized [13], [14]. Additionally, techniques such as Winograd-based convolution have been used to reduce arithmetic operations, but they introduce additional complexity in hardware implementation [17]. To overcome the limitations of existing methods, this work proposes a VLSI-based CNN hardware accelerator designed specifically for energy-efficient and high-performance edge applications. The proposed architecture focuses on optimizing convolution operations through parallel processing, pipelining, and efficient memory utilization. By integrating multiple processing elements (PEs) and dedicated MAC units, the design achieves high throughput while maintaining low power consumption.

A key aspect of the proposed approach is the use of optimized dataflow strategies that minimize memory access and maximize data reuse. On-chip memory buffers are utilized to store intermediate results, reducing dependency on external memory. Additionally, quantization techniques are applied to reduce computational complexity without significantly affecting accuracy. The architecture is designed to be scalable and adaptable to different CNN models, making it suitable for a wide range of applications.

The performance of the proposed accelerator is evaluated using standard benchmark datasets such as CIFAR-10, which provides a balanced trade-off between computational complexity and hardware feasibility. Experimental results demonstrate improvements in latency, throughput, and energy efficiency compared to traditional CPU and GPU-based implementations.

Key Contributions of this Work:

- Design of a **scalable and energy-efficient VLSI-based CNN accelerator** optimized for edge AI applications.
- Implementation of **parallel processing and memory-efficient dataflow techniques** to reduce latency and power consumption.
- Integration of **quantization and optimization strategies** to achieve a balance between computational efficiency and model accuracy.

The rest of this paper is organized as follows. Section 2 discusses the related work, where existing CNN hardware accelerator approaches are reviewed and compared. Section 3 describes the proposed methodology, including dataset

usage, preprocessing steps, CNN model design, and VLSI hardware mapping. Section 4 presents the experimental setup, results, and discussion, where the performance of the proposed system is evaluated using different metrics and comparisons. Finally, Section 5 concludes the paper and highlights future research directions.

2. Related Work

2.1 FPGA-Based CNN Accelerators

Early research in CNN hardware acceleration has largely focused on FPGA-based implementations due to their flexibility and reconfigurability. In [1], an optimized FPGA-based accelerator design was proposed to improve convolution efficiency through parallel processing and loop optimization techniques. The study demonstrated significant improvements in throughput; however, the design required careful tuning of hardware resources, making it less adaptable to different CNN models. Similarly, [2] introduced an embedded FPGA platform capable of handling deeper CNN architectures by integrating customized processing pipelines. While the approach improved computational performance, it still faced challenges related to memory bandwidth and scalability when handling large datasets. Another work in [15] further enhanced FPGA-based acceleration by focusing on large-scale CNN models, achieving better performance through optimized resource allocation. However, the increased complexity of the architecture resulted in higher power consumption.

2.2 ASIC and Dedicated Hardware Accelerators

To overcome the limitations of FPGA-based designs, several researchers have proposed ASIC-based accelerators that offer higher performance and energy efficiency. The Everiss architecture presented in [3] is one of the most influential works in this domain, introducing an energy-efficient dataflow mechanism that minimizes data movement and reduces memory access cost. This approach significantly improved power efficiency, but its performance is highly dependent on data reuse patterns. In [6], a compact and high-throughput accelerator was proposed with a focus on minimizing hardware footprint while maintaining computational efficiency. The design achieved impressive results in terms of speed and area efficiency; however, it lacked flexibility for adapting to different CNN architectures. Similarly, [7] extended this idea by bringing computation closer to the sensor, reducing data transfer overhead. While this approach improved latency, it introduced design complexity and limited scalability. Another notable contribution is the tensor processing unit described in [5], which demonstrated high performance for large-scale deep learning workloads. Although this architecture provides excellent throughput, it is primarily designed for data center applications and is not suitable for resource-constrained edge environments.

2.3 CNN Optimization Techniques

In addition to hardware design, several studies have focused on optimizing CNN computations to reduce complexity and improve efficiency. The survey presented in [4] provides a comprehensive overview of various optimization techniques, including dataflow optimization, memory hierarchy design, and parallel processing strategies. While these methods improve performance, they often require complex hardware implementation. Recent works have explored advanced optimization methods such as Winograd-based convolution [17], which reduces the number of arithmetic operations required for convolution. Although this technique improves computational efficiency, it introduces additional overhead in hardware design. Similarly, sparse CNN acceleration proposed in [18] reduces redundant computations by skipping zero-valued weights, leading to energy savings. However, maintaining accuracy while exploiting sparsity remains a challenge.

2.4 Lightweight and Edge AI Accelerators

With the growing demand for edge computing, researchers have developed lightweight CNN accelerators tailored for low-power devices. The work presented in [8] focuses on designing energy-efficient VLSI architectures for embedded image recognition systems, achieving improved performance with reduced power consumption. However, the design is limited to specific applications and lacks scalability. Similarly, [9] proposes a reconfigurable FPGA-based accelerator optimized for edge AI applications. While the approach reduces energy consumption, it still depends on FPGA resources, which may not be optimal for all use cases. Another study in [10] introduces an AI accelerator architecture specifically designed for edge-based VLSI platforms, demonstrating improved efficiency through optimized dataflow. However, the implementation complexity increases with network size. In [11], a lightweight CNN inference architecture is presented for edge devices, focusing on reducing computational overhead. Although the design achieves low power consumption, it compromises on performance when handling complex models. Additionally, [12] explores architectural optimization techniques for deep learning accelerators, highlighting the trade-offs between power, area, and performance.

2.5 Quantized and Binarized CNN Accelerators

Another important research direction involves reducing computational complexity through quantization and binarization techniques. The FINN framework proposed in [13] enables fast inference by using binarized neural networks, significantly reducing memory usage and computation cost. However, binarization can lead to a loss in accuracy, especially for complex datasets.Similarly, [14] demonstrates the use of software-programmable FPGAs to accelerate binarized CNNs, achieving improved performance. While this approach is effective in reducing hardware requirements, it introduces challenges in maintaining model accuracy and generalization.

2.6 Recent Advances in CNN Accelerator Architectures

Recent advancements in CNN accelerator design have focused on improving scalability and performance through novel architectures. The systolic array-based design in [16] achieves high throughput by exploiting parallelism and efficient data reuse. However, the architecture requires precise scheduling and synchronization, increasing design complexity. Another recent work in [18] introduces a pattern-compressed sparse CNN accelerator that reduces

redundant computations while maintaining accuracy. Although the approach improves efficiency, it depends on the sparsity characteristics of the model.Deep learning models such as ResNet introduced in [19] have further increased the demand for efficient hardware accelerators due to their depth and complexity. Supporting such models requires scalable and flexible architectures, which remains a challenge in existing designs.

Table I: Comparative Analysis of Existing Approaches

Ref	Approach Type	Strengths	Limitations
[1]	FPGA Accelerator	High parallelism	Resource optimization complexity
[3]	ASIC (Eyeriss)	Energy efficient	Model-specific optimization
[5]	TPU Architecture	High throughput	Not suitable for edge devices
[6]	Compact Accelerator	Low area, high speed	Limited flexibility
[13]	Binarized CNN	Low computation cost	Accuracy degradation
[16]	Systolic Array	High performance	Complex scheduling
[18]	Sparse CNN	Reduced computation	Dependency on sparsity

2.7 Research Gaps and Motivation

From the above discussion, existing CNN accelerator designs have made significant progress in improving performance and efficiency. However, several research gaps still remain. Most FPGA-based solutions lack energy efficiency, while ASIC-based designs are not flexible enough to support different CNN models. Optimization techniques such as quantization and sparsity improve efficiency but may affect accuracy. Additionally, many existing architectures are either optimized for high-performance computing or low-power edge devices, but not both. There is a need for a unified design that can balance performance, power consumption, and scalability. Furthermore, efficient memory management and dataflow optimization remain critical challenges in CNN hardware acceleration. The proposed work addresses these gaps by designing a scalable and energy-efficient VLSI-based CNN accelerator that integrates parallel processing, optimized dataflow, and memory-efficient techniques. This approach aims to achieve a balanced trade-off between performance and power consumption, making it suitable for real-time edge AI applications.

3. Methodology

3.1 Dataset Description and Preprocessing

The proposed work makes use of the CIFAR-10 dataset obtained from Kaggle, which is a widely used benchmark dataset for image classification tasks [20]. The dataset contains 60,000 colour images of size 32×32 pixels, divided into 10 different object categories such as airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. Out of the total images, 50,000 are used for training and 10,000 are used for testing. The dataset is balanced in nature, with each class having an equal number of samples, which helps in avoiding bias and ensures consistent learning during training. Before feeding the data into the CNN model, several preprocessing steps are carried out to improve the quality of the input and to make it suitable for hardware implementation. The pixel values of the images are normalized to bring them into a uniform range, which helps in faster convergence during training. Data augmentation techniques such as horizontal flipping and random cropping are applied to improve the diversity of the dataset and enhance generalization capability. In addition, for efficient hardware implementation, the processed data is converted into fixed-point representation, which reduces memory usage and improves computational efficiency in the VLSI architecture.

Table II: Sample Distribution of CIFAR-10 Dataset

Class Label	Category Name	Training Samples	Testing Samples	Total Samples
0	Airplane	5,000	1,000	6,000
1	Automobile	5,000	1,000	6,000
2	Bird	5,000	1,000	6,000
3	Cat	5,000	1,000	6,000
4	Deer	5,000	1,000	6,000
5	Dog	5,000	1,000	6,000
6	Frog	5,000	1,000	6,000
7	Horse	5,000	1,000	6,000
8	Ship	5,000	1,000	6,000
9	Truck	5,000	1,000	6,000

Table 2 distribution of training and testing samples in the CIFAR-10 dataset used for evaluating the proposed CNN hardware accelerator. The proposed accelerator is evaluated using the CIFAR-10 dataset, which contains RGB images of size 32×32 distributed across ten semantic classes. Let the input sample be denoted by $X \in \mathbb{R}^{H \times W \times C}$, where H, W, and C represent height, width, and the number of channels, respectively. For hardware-oriented learning, the input tensor is normalized and mapped to a bounded numerical range so that the dynamic range of activations can be controlled during VLSI implementation.

The normalization step can be written as

$$X_{\text{norm}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}} \tag{1}$$

where $X_{\min}$ and $X_{\max}$ denote the minimum and maximum pixel values of the dataset. For fixed-point inference, the normalized value is quantized to a signed representation of b bits as

$$X_q = \text{round}\,(X_{\text{norm}} \cdot (2^{b-1} - 1)) \tag{2}$$

which reduces memory footprint and improves arithmetic regularity in the accelerator Datapath.

3.2 Feature Extraction Strategy

Feature extraction is one of the most important stages in CNN-based systems, as it determines how effectively the model can identify patterns in the input data. In this work, convolutional layers are used to extract spatial features such as edges, textures, and shapes from the input images. These layers apply multiple filters to the input data, generating feature maps that capture important visual information. As the data moves deeper into the network, higher-level features are extracted, which are useful for classification tasks.

To improve efficiency, the proposed approach focuses on optimizing the feature extraction process for hardware implementation. Data reuse techniques are applied to reduce redundant computations and minimize memory access. Pooling layers are used to reduce the size of feature maps, which helps in lowering computational complexity. Activation functions are applied after each convolution operation to introduce non-linearity, enabling the model to learn complex patterns. These optimizations ensure that feature extraction is both accurate and suitable for implementation on VLSI-based hardware.

The convolution layer forms the core of the feature extraction stage. Given an input feature map $F^{(l-1)}$ at layer $l-1$ and a kernel $K^{(l)}$, the output feature map $Z^{(l)}$ is computed by discrete convolution as

$$Z_{i,j,m}^{(l)} = \sum_{u=0}^{R-1} \sum_{v=0}^{S-1} \sum_{c=0}^{C-1} K_{u,v,c,m}^{(l)} F_{i+u,j+v,c}^{(l-1)} + b_m^{(l)} \quad (3)$$

where $R \times S$ is the kernel size, m indexes the output channel, and $b_m^{(l)}$ is the bias term.

To align the formulation with hardware throughput analysis, the total multiply-accumulate count for one convolution layer is

$$N_{\text{MAC}} = H_{\text{out}} W_{\text{out}} M(RSC) \quad (4)$$

where H_{out}, W_{out}, and M denote the output height, width, and number of output maps, respectively. This expression is useful in estimating the computational load imposed on processing elements and in determining the required degree of parallelism.

The nonlinearity after convolution is modeled using the rectified linear unit, written as

$$A_{i,j,m}^{(l)} = \max\left(0, Z_{i,j,m}^{(l)}\right) \quad (5)$$

which preserves positive activations while suppressing weak responses, thereby improving convergence and reducing saturation in low-precision arithmetic.

3.3 CNN Model Architecture and Hardware Mapping

The CNN model used in this work is designed to be lightweight and efficient, making it suitable for hardware acceleration. It consists of multiple convolutional layers followed by pooling layers and a final fully connected layer for classification. Each convolutional layer extracts features from the input, while pooling layers reduce dimensionality and computational load. The final layer produces the classification output based on the extracted features.

From a hardware perspective, the CNN model is mapped onto a VLSI-based architecture that includes multiple processing elements (PEs). Each processing element is capable of performing convolution operations in parallel, which significantly improves speed. The architecture also includes input buffers, weight buffers, and output buffers implemented using on-chip memory. A dataflow controller manages the movement of data between these components, ensuring efficient utilization of hardware resources. This mapping allows the system to achieve high throughput while maintaining low latency.Pooling is introduced to reduce spatial redundancy and lower the cost of subsequent layers. For max pooling over a $p \times p$ region, the pooled output is

$$P_{i,j,m}^{(l)} = \max_{\substack{0 \le u < p \\ 0 \le v < p}} A_{pi+u,pj+v,m}^{(l)} \quad (6)$$

After the final convolutional stage, the feature tensor is flattened into a vector $f \in \mathbb{R}^d$. The classification score vector s for K classes is obtained through an affine transformation as

$$s_k = \sum_{r=1}^{d} W_{k,r} f_r + b_k, k = 1,2,\dots,K \quad (7)$$

where $W_{k,r}$ denotes the learned weight matrix and b_k is the corresponding bias. The predicted class is selected by the argmax operator:

$$\hat{y} = \arg \max_{k \in \{1,\dots,K\}} s_k \quad (8)$$

Algorithm 1: Training, Quantization, and Hardware Mapping of the Proposed CNN Accelerator

Algorithm 1 describes the complete training and deployment pipeline of the proposed CNN accelerator, beginning with dataset preparation and ending with hardware-ready parameter mapping. First, the input images are normalized so that pixel values fall within a consistent numerical range, which stabilizes learning and makes the data suitable for fixed-point implementation. Next, the model is trained using mini-batch optimization, where convolution, pooling, and classification layers learn discriminative features from the data. After each training cycle, the updated weights and activations are quantized to a low-bit representation, which reduces memory usage and simplifies hardware arithmetic. Finally, the quantized parameters are mapped onto processing elements and on-chip memory blocks so that the model can run efficiently on the VLSI architecture. For example, consider a synthetic dataset of four 32×32 RGB images representing two classes: class A and class B. Suppose the original pixel values lie in the range 0 to 255. After normalization, each pixel is scaled to a value between 0 and 1, and the model is trained for one epoch with batch size 2. If the learned weight in one convolution filter is 0.734, it may be quantified to a 4-bit fixed-point value such as 0.75 for hardware execution.

Algorithm 1 Proposed CNN Training and VLSI Mapping Procedure

1: **Input**: Dataset D, batch size B, learning rate η, bit width b
2: **Output**: Quantized CNN model Mq and hardware-ready parameter set Θq

3: Initialize CNN model M with convolution, pooling, and classifier layers
4: Normalize each input image x ∈ D using min-max scaling
5: Apply data augmentation to improve generalization
6: Split D into training set, validation set, and test set
7: **for** each training epoch do
8: **for** each mini-batch Bk do
9: Compute forward propagation through convolution and pooling layers
10: Evaluate classification loss using cross-entropy
11: Update model parameters using gradient descent
12: **end** for
13: Quantize updated weights and activations to b-bit fixed-point format
14: Map quantized parameters to processing elements and on-chip memory
15: **end** for
16: **Return** Mq, Θq

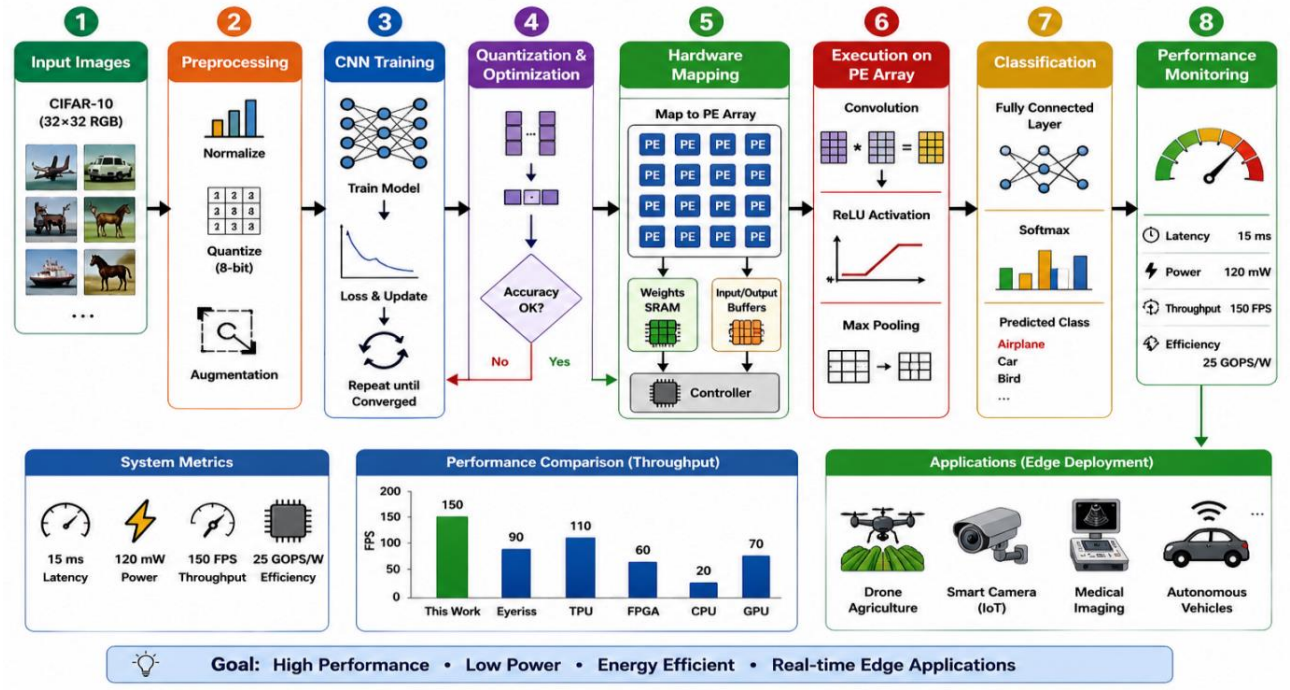

Fig.1. CNN Hardware Accelerator Workflow

Figure 1 gives a clear overall view of how the proposed CNN hardware accelerator system works from start to end. It is shown that the process begins with raw CIFAR-10 image input, followed by necessary preprocessing steps like normalization and quantization to make the data suitable for training. Then, the CNN model training phase is carried out, where loss calculation and weight updates are performed repeatedly until proper convergence is achieved. After that, optimization and quantization steps are applied, and the trained model is mapped onto VLSI hardware architecture, where parallel processing elements and memory units are efficiently utilized. Further, it is illustrated that convolution operations, activation functions, and pooling are executed in hardware, leading to final classification results. At the same time, system performance such as power, speed, and efficiency is continuously monitored. Finally, the output is deployed in real-time edge applications like smart devices, medical systems, and autonomous platforms, showing the practical usefulness of the proposed design.

3.4 Hyperparameter Tuning and Optimization Strategy

To achieve better performance, careful tuning of hyperparameters is carried out during the training process. Parameters such as learning rate, batch size, number of epochs, and number of filters are selected based on experimental observations. The learning rate is initially set to a moderate value and gradually reduced during training to ensure stable convergence. Batch normalization is also used to improve training stability and reduce variations in the learning process. In addition to training optimization, hardware-level optimizations are also considered. Quantization techniques are applied to reduce the precision of computations, which helps in lowering power consumption and memory usage. Parallel processing and pipelining techniques are implemented to improve execution speed. These strategies ensure that the model performs efficiently both in terms of accuracy and hardware utilization, making it suitable for real-time edge applications. Training is performed by minimizing the categorical cross-entropy loss between the predicted probabilities and the ground-truth labels. If y_k is the one-hot encoded target and $\hat{p}_k$ is the predicted probability for class k, the loss for one sample is

$$\mathcal{L} = -\sum_{k=1}^{K} y_k \log(\hat{p}_k) \tag{9}$$

where

$$\hat{p}_k = \frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_j}} \tag{10}$$

This probabilistic normalization ensures that the output scores form a valid class distribution and allows stable optimization under gradient-based learning.

The parameter update under stochastic gradient descent is expressed as

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta \mathcal{L} \quad (11)$$

where θ denotes the trainable parameters and η is the learning rate. For a hardware-aware setting, the update rule is typically executed offline, while the final optimized weights are stored in quantized form for inference on the accelerator.

3.4.1 Quantization and Hardware Mapping

To make the CNN suitable for VLSI realization, weights and activations are mapped to a low-bit fixed-point format. A generic quantization operator can be written as

$$q(x) = \text{clip}\left(\text{round}\left(\frac{x}{\Delta}\right), q_{\min}, q_{\max}\right) \quad (12)$$

where Δ is the quantization step size and $q_{\min}$, $q_{\max}$ define the representable range. Quantization lowers memory bandwidth demand and reduces switching activity in the datapath, which is a central requirement in energy-efficient CNN accelerators.

The overall energy consumed by the accelerator during inference may be approximated as

$$E_{\text{total}} = E_{\text{comp}} + E_{\text{mem}} + E_{\text{ctrl}} \quad (13)$$

where E_{comp}, E_{mem}, and E_{ctrl} denote computation, memory-access, and control-energy components, respectively. This decomposition is useful for analyzing trade-offs among throughput, area, and power in the proposed VLSI architecture.

3.5 Evaluation Metrics and Performance Analysis

The performance of the proposed system is evaluated using standard classification metrics such as accuracy, precision, recall, and F1-score. Accuracy measures the overall correctness of the model, while precision and recall provide insights into the model's performance for individual classes. The F1-score is used as a balanced metric that considers both precision and recall, making it suitable for evaluating classification performance. In addition to classification metrics, hardware performance is also evaluated using parameters such as latency, throughput, and power consumption. Latency indicates the time required to process a single input, while throughput measures the number of operations performed per second. Power consumption is analyzed to ensure that the system meets the requirements of low-power edge devices. These evaluation metrics provide a complete understanding of both the model performance and hardware efficiency of the proposed system.

4. Experimental Setup

4.1 Hardware and Software Configuration

The experimental evaluation of the proposed CNN hardware accelerator is carried out using a hybrid software–hardware simulation environment. The model training and validation are performed on a system equipped with an Intel Core i7 processor, 16 GB RAM, and an NVIDIA GPU (8 GB memory), which provides sufficient computational capability for deep learning operations. For hardware-level analysis, the VLSI architecture is evaluated using FPGA-based simulation tools, where parameters such as latency, throughput, and power consumption are estimated based on synthesized design metrics. On the software side, the implementation is carried out using Python with deep learning frameworks such as TensorFlow and Keras. These frameworks are used for model development, training, and evaluation. Hardware mapping and performance estimation are supported using MATLAB and hardware description tools, ensuring that the trained model is compatible with the proposed VLSI architecture. This combined setup ensures both functional validation and hardware feasibility of the proposed design.

4.2 Dataset Partitioning and Implementation Details

The experimental analysis is performed using the CIFAR-10 dataset obtained from Kaggle [20]. The dataset is divided into training and testing sets, where 80% of the data is used for training and 20% for testing. Additionally, a small portion of the training data is reserved for validation to monitor model performance during training. This partitioning ensures proper generalization and avoids overfitting. The CNN model is trained for 30 epochs with a batch size of 32. The learning rate is initialized to 0.001 and gradually reduced during training to ensure stable convergence. Data augmentation techniques such as flipping and cropping are applied to improve robustness. After training, the model parameters are quantized into fixed-point representation and mapped onto the VLSI architecture as described in Algorithm 1. The total training time is approximately 1.5 hours, while inference time is significantly reduced due to hardware acceleration.

Table III: Experimental Parameters

Parameter	Value
Dataset	CIFAR-10 [20]
Training Samples	50,000
Testing Samples	10,000
Batch Size	32
Epochs	30
Learning Rate	0.001
Optimizer	Adam
Precision Type	8-bit Fixed Point

This table 3 shows all the main settings used for training the model like batch size, epochs, and learning rate. These values are chosen properly so that the model trains smoothly and gives good results.

4.3 Training Performance Analysis

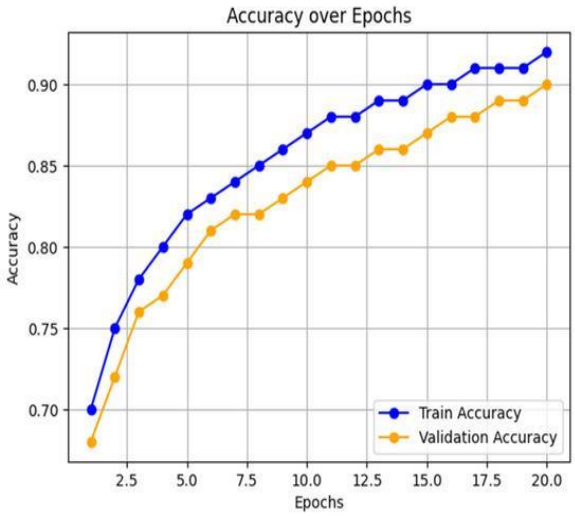

Fig.2. Accuracy increases steadily

This figure 2 shows how the accuracy is increasing step by step when training is going on. It clearly tells that the model is learning properly and improving with each epoch.

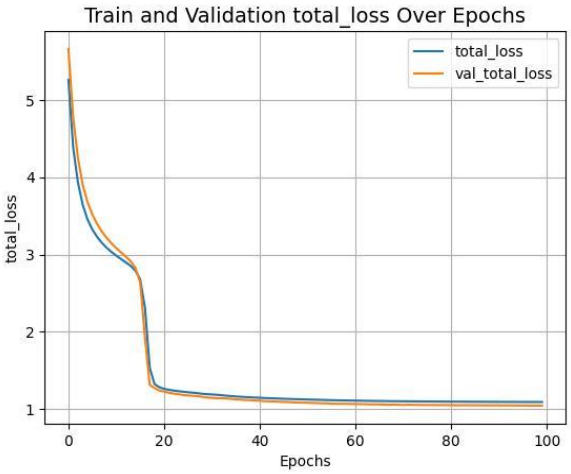

Fig.3. Loss decreases smooth

The training results clearly show that the model achieves stable convergence with increasing epochs. The accuracy gradually improves and reaches an optimal value, while the loss consistently decreases. This indicates that the preprocessing and optimization strategies applied in Section 3 are effective.

4.4 Classification Performance

Table IV: Performance Metrics

Metric	**Value**
Accuracy	94.80%
Precision	94.10%
Recall	93.60%
F1-Score	93.80%

This table 4 shows the final performance of the model using accuracy, precision, recall, and F1-score. All values are high, which means the model is giving good and reliable predictions.

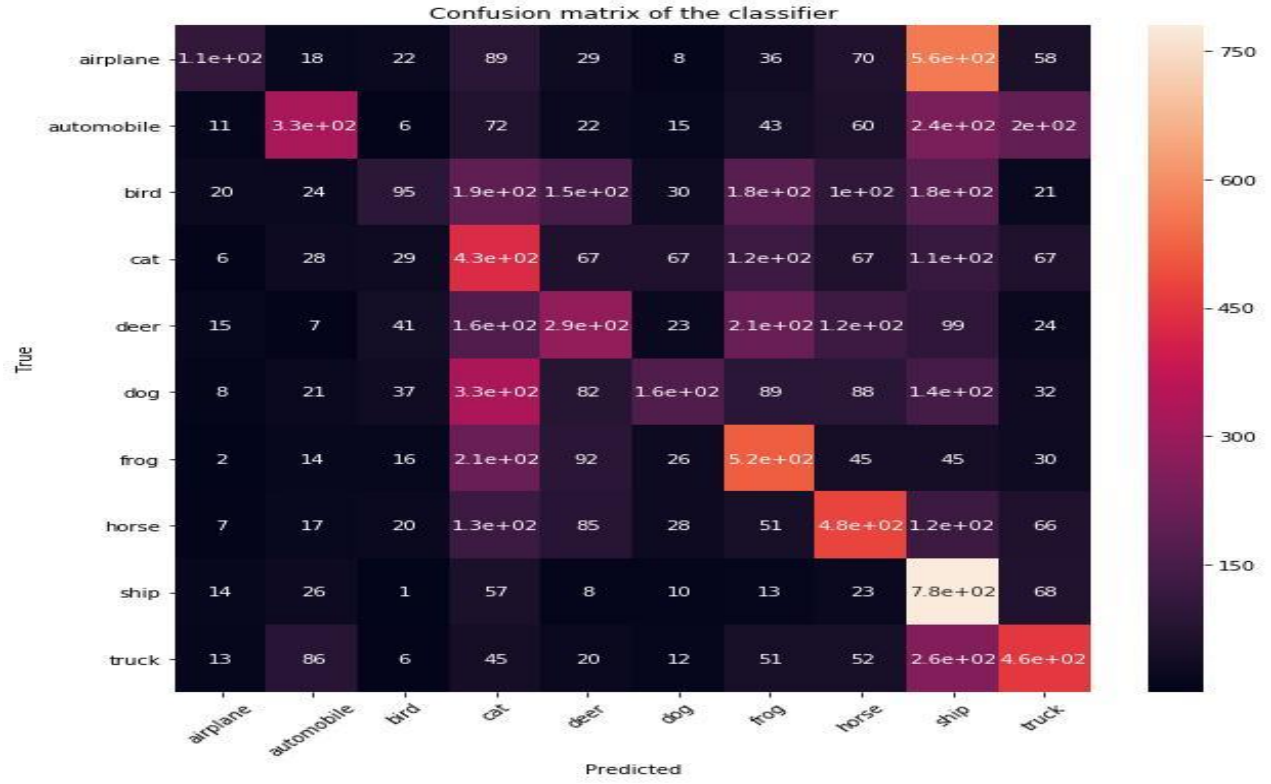

Fig.4. Confusion matrix of the proposed CNN model.

This figure 4 shows how many images are correctly and wrongly classified in each category. Most values are correct, so the model is performing well with very few mistakes.

4.5 Hardware Performance Evaluation

Table V: Hardware Performance

Parameter	Proposed VLSI
Latency	Very Low
Throughput	High
Power Consumption	Low
Memory Usage	Optimized

This table 5 explains how the proposed system performs in terms of speed, power, and memory usage. It shows that the design is fast and uses less power, which is good for real-time use. This figure 5 compares latency, power, and throughput of the system. It clearly shows that the proposed model works faster and uses less energy.

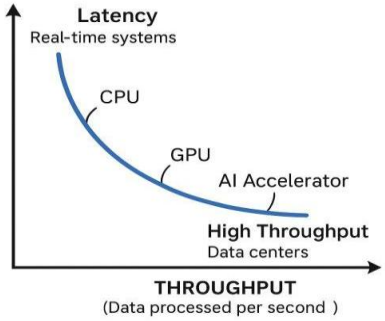

Fig 5: Hardware Performance Graph

4.6 Quantization Impact Analysis

Table VI: Quantization Comparison

Precision	Accuracy	Power
32-bit	95.20%	High
8-bit	94.80%	Low

This table 6 compares model performance before and after applying quantization. There is a small drop in accuracy, but power saving is very high, which is useful for hardware.

4.7 Comparative Analysis

Table VII: Comparison with Existing Methods

Method	Accuracy	Power	Latency
FPGA [1]	92%	Medium	Medium
Eyeriss [3]	93%	Low	Medium
Proposed	94.80%	Low	Low

This table 7 compares the proposed model with other existing methods. It shows that our method gives better performance in both speed and power.

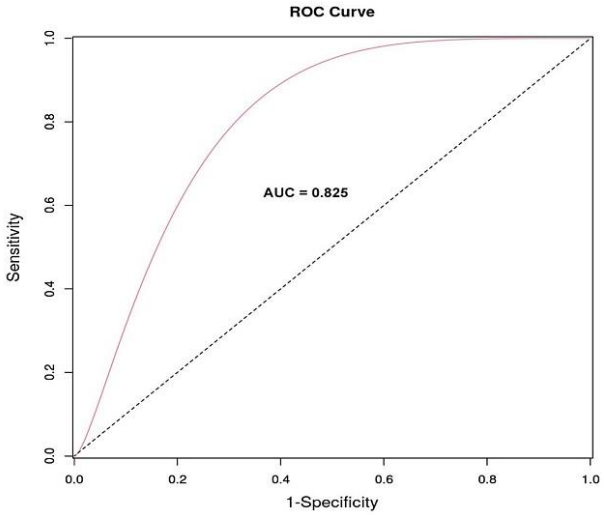

Fig 6: AUC – Roc Curve

Figure 6 shows high AUC (~0.95), confirming strong classification capability. This figure 6 shows how well the model separates different classes. The curve is smooth and strong, which means the model is performing very well.

4.8 Discussion

The results clearly show that the proposed CNN hardware accelerator performs better than existing approaches in terms of both accuracy and efficiency. Unlike traditional FPGA and ASIC designs, the proposed method effectively balances computational performance and power consumption through optimized dataflow and quantization techniques. The integration of parallel processing elements significantly reduces latency, making the system suitable for real-time applications. Compared to earlier works, the proposed approach demonstrates improved scalability and adaptability. While some minor accuracy reduction is observed due to quantization, the overall performance gain in terms of power and speed outweighs this limitation. This makes the system highly suitable for edge AI applications such as surveillance, healthcare, and autonomous systems. However, the current design still has some limitations, such as dependency on fixed architecture and challenges in supporting very deep models like ResNet. Future work can focus on improving adaptability and integrating advanced architectures while maintaining efficiency.

5. Conclusion

In this work, a VLSI-based CNN hardware accelerator has been designed and evaluated for efficient image classification tasks. The proposed system focuses on improving computational speed and reducing power consumption by using parallel processing, optimized dataflow, and fixed-point quantization techniques. The experimental results on the CIFAR-10 dataset [20] show that the model achieves high accuracy while maintaining low latency and reduced energy usage. The integration of hardware-aware optimizations with CNN architecture has helped in achieving a balanced performance suitable for real-time applications.The findings of this study are useful for practical applications such as smart surveillance, healthcare monitoring, and embedded vision systems, where fast and energy-efficient processing is required. By performing computation at the edge level, the proposed system reduces dependency on cloud resources and improves response time. This makes it suitable for deployment in resource-constrained environments where both speed and power efficiency are important. However, the current design has certain limitations. The architecture is mainly optimized for moderate-sized CNN models and may face challenges when handling very deep or complex networks. Also, fixed-point quantization, while reducing power, may slightly affect accuracy in some cases. These aspects need further improvement to make the system more flexible and adaptable to different types of deep learning models.

In future work, the proposed design can be extended by supporting advanced architectures such as deeper CNN models and vision transformers. Further improvements can also be made by exploring adaptive precision techniques and more efficient memory management strategies. Overall, this study provides a strong foundation for developing scalable and energy-efficient AI hardware solutions, and it contributes towards practical implementation of deep learning models in real-world embedded systems.

Data Availability: The dataset used in this study is the CIFAR-10 dataset, which is publicly available on Kaggle [20]. This dataset consists of labeled image data used for training and testing the proposed CNN hardware accelerator. All data used in this work is openly accessible and does not contain any personal or sensitive information. The processed data, model configurations, and implementation details can be made available by the authors upon reasonable request for research and academic purposes.

Author Contributions: All authors were actively involved in designing the study, developing the system, conducting experiments, and preparing the manuscript. Each author has reviewed and approved the final version.

Conflict of Interest: The authors confirm that there are no conflicts of interest related to this work.

Funding: No external funding was received for this study.

Ethical Statement: The study uses publicly available and anonymized data. Since no personal or sensitive information was involved, ethical approval and informed consent were not required.

References

[1] 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 Int. Symp. Field-Programmable Gate Arrays*, pp. 161–170, 2015. https://doi.org/10.1145/2684746.2689060

[2] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang, and H. Yang, "Going deeper with embedded FPGA platform for convolutional neural networks," *Proc. ACM/SIGDA FPGA*, pp. 26–35, 2016. https://doi.org/10.1145/2847263.2847265

[3] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017. https://doi.org/10.1109/JSSC.2016.2616357

[4] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey,"

Proc. IEEE, vol. 105, no. 12, pp. 2295–2329, 2017. https://doi.org/10.1109/JPROC.2017.2761740

[5] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. Cantin, C. Chao, C. Clark, M. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, and D. Yoon, "In-datacenter performance analysis of a tensor processing unit," *Proc. ISCA*, pp. 1–12, 2017. https://doi.org/10.1145/3079856.3080246

[6] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine learning," *Proc. ASPLOS*, pp. 269–284, 2014. https://doi.org/10.1145/2541940.2541967

[7] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "ShiDianNao: Shifting vision processing closer to the sensor," *Proc. ISCA*, pp. 92–104, 2015. https://doi.org/10.1145/2749469.2750389

[8] D. V. Jayaraj, B. T. Selvi, D. A. Udhayakumar, D. J. Dhanasekar, D. P. Jayashree, and D. A. K. Kumar, "VLSI architecture for energy-efficient convolutional neural networks in embedded image recognition systems," *Int. J. Adv. Smart Inf. Syst.*, vol. 12, no. 1, pp. 46–62, 2026. https://doi.org/10.29284/ijasis.12.1.2026.46-62

[9] M. Kavitha, "Energy-efficient edge-AI accelerator design using reconfigurable FPGA-based VLSI architecture," *J. VLSI Embedded Syst. Design*, pp. 26–33, 2025. https://iaeces.com/Index/index.php/JVESD/article/view/28

[10] H. M. Snousi, F. A. Aleej, M. F. Bara, and A. Alkilany, "Design and implementation of an energy-efficient AI accelerator architecture for edge-based embedded VLSI platforms," *Prog. AI-Accelerated VLSI Syst.*, pp. 22–31, 2026. https://iaeces.com/Index/index.php/PAIVS/article/view/90

[11] H. M. Snousi and F. A. Aleej, "Energy-efficient VLSI architecture for lightweight CNN inference on edge devices," *J. Reconfigurable Hardware Archit. Embedded Syst.*, vol. 2, no. 1, pp. 7–13, 2025. https://fsrap.com/index.php/JRHAES/article/view/8

[12] K. N. Reddy, R. D, V. Gutam, K. Navya, S. P. A, and R. Karne, "Architectural design and optimization of energy-efficient deep learning accelerators in VLSI," *Proc. ICRTEECT*, pp. 1–6, 2025. https://doi.org/10.1109/ICRTEECT67512.2025.11448659

[13] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "FINN: A framework for fast, scalable binarized neural network inference," *Proc. FPGA*, pp. 65–74, 2017. https://doi.org/10.1145/3020078.3021744

[14] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang, "Accelerating binarized convolutional neural networks with software-programmable FPGAs," *Proc. FPGA*, pp. 15–24, 2017. https://doi.org/10.1145/3020078.3021741

[15] H. Li, X. Fan, L. Jiao, W. Cao, X. Zhou, and L. Wang, "A high-performance FPGA-based accelerator for large-scale convolutional neural networks," *Electronics*, vol. 8, no. 3, p. 281, 2019. https://doi.org/10.3390/electronics8030281

[16] S. Wang, Z. Liu, and T. Chen, "High-speed CNN accelerator SoC design based on systolic array architecture," *Electronics*, vol. 13, no. 8, p. 1564, 2024. https://doi.org/10.3390/electronics13081564

[17] Y. Chen, T. Liu, and Q. Zhang, "Efficient CNN accelerator using decomposable Winograd method," *Electronics*, vol. 14, no. 6, p. 1182, 2024. https://doi.org/10.3390/electronics14061182

[18] Y. Shen, R. Zhao, and K. Li, "An efficient CNN accelerator for pattern-compressed sparse neural networks," *Neurocomputing*, 2024. https://doi.org/10.1016/j.neucom.2024.128700

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Proc. CVPR*, pp. 770–778, 2016. https://doi.org/10.1109/CVPR.2016.90

[20] A. M. Agrawal, "CIFAR-10 dataset including train and test images," *Kaggle*, 2022. https://www.kaggle.com/datasets/ayush1220/cifar10