A Novel Implementation of CNN on FPGA-Based Embedded System

By {your name}, {your affiliation}, {your email}

# Abstract

Convolutional neural networks (CNNs) are widely used for various computer vision tasks, such as image classification, object detection, and face recognition. However, CNNs are computationally intensive and require a large amount of memory and power consumption, which limits their deployment on resource-constrained embedded systems. In this paper, we propose a novel implementation of CNN on a field-programmable gate array (FPGA)-based embedded system, which achieves high performance, low power, and low memory consumption. We design a custom hardware architecture that exploits the parallelism and sparsity of CNNs and implements various optimization techniques, such as quantization, pruning, and compression. We also develop a software framework that automatically generates the FPGA configuration and the embedded software from a given CNN model. We evaluate our implementation on several benchmark datasets and CNN models and show that it outperforms the state-of-the-art FPGA-based CNN implementations in accuracy, speed, power, and memory efficiency.

KEYWORDS: Convolutional Neural Networks (CNNs), FPGA-based Embedded System, Custom Hardware Architecture, Optimization Techniques, Quantization, Pruning, Compression, Software Framework, Benchmark Datasets, CNN Models, Accuracy, Performance, Power Efficiency, Memory Efficiency

# Introduction

Convolutional neural networks (CNNs) are a type of deep neural networks that have achieved remarkable results in various computer vision tasks, such as image classification [1], object detection [2], and face recognition [3]. CNNs consist of multiple layers of neurons that perform convolution, pooling, activation, and fully connected operations on the input data. CNNs can learn complex and high-level features from the data and achieve superior accuracy compared to traditional machine learning methods. However, CNNs have drawbacks, such as high computational complexity, large memory footprint, and high power consumption. These drawbacks pose significant challenges for deploying CNNs on resource-constrained embedded systems, such as smartphones, drones, and wearable devices, which have limited hardware resources, battery life, and cooling capacity. Therefore, there is a need for efficient and effective implementation of CNNs on embedded systems, which can balance the trade-off between accuracy, performance, power, and memory.

Field-programmable gate arrays (FPGAs) are reconfigurable hardware devices that can implement custom logic circuits on various programmable logic blocks and interconnects. FPGAs have several advantages for implementing CNNs on embedded systems, such as high parallelism, flexibility, scalability, and reconfigurability. FPGAs can exploit the parallelism of CNNs by mapping multiple neurons and operations to different logic blocks and performing them concurrently. FPGAs can also adapt to different CNN models and datasets by reconfiguring the logic blocks and interconnects according to the network structure and parameters. FPGAs can also scale up or down the hardware resources according to the application's performance and power requirements. Moreover, FPGAs can be reconfigured at run-time, which enables dynamic adaptation and optimization of the CNN implementation. Therefore, FPGAs are a promising platform for implementing CNNs on embedded systems.

However, implementing CNNs on FPGAs also faces challenges, such as limited hardware resources, memory bandwidth, and design complexity. FPGAs have limited logic blocks, memory blocks, and interconnects, which restrict the number and size of CNN layers that can be implemented on a single FPGA device. FPGAs also have limited memory bandwidth, which limits the data transfer rate between the FPGA and the external memory, such as DRAM. This can cause performance degradation and power overhead for CNNs with significant input and output data. Furthermore, implementing CNNs on FPGAs requires high hardware design expertise and software-hardware co-design skills, which are rare among CNN developers and users. Therefore, there is a need for novel and efficient hardware architectures and software frameworks for implementing CNNs on FPGAs, which can overcome these challenges and achieve high accuracy, performance, power, and memory efficiency.

This paper proposes a novel implementation of CNN on an FPGA-based embedded system, which addresses the above challenges and achieves high accuracy, performance, power, and memory efficiency. Our main contributions are as follows:

* We design a custom hardware architecture that exploits the parallelism and sparsity of CNNs and implements various optimization techniques, such as quantization, pruning, and compression. Our hardware architecture consists of a reconfigurable array of processing elements (PEs) that perform convolution, pooling, and activation operations and a controller that manages the data flow and synchronization. Our hardware architecture can support different CNN models and datasets by reconfiguring the number and type of PEs and the interconnection network. Our hardware architecture can also reduce the computation, memory, and power consumption of CNNs by applying quantization, pruning, and compression techniques, which reduce the bit-width, number, and size of the network parameters and data.
* We develop a software framework that automatically generates the FPGA configuration and the embedded software from a given CNN model. Our software framework takes a CNN model as input and performs various steps, such as network analysis, hardware synthesis, software compilation, and FPGA programming. Our software framework can analyze the network structure and parameters and select the optimal hardware configuration and optimization techniques for the target FPGA device and embedded system. Our software framework can also synthesize the hardware architecture, compile the embedded software, and program the FPGA device without requiring manual intervention or hardware design expertise from the user.
* We evaluate our implementation on several benchmark datasets and CNN models and show that it outperforms the state-of-the-art FPGA-based CNN implementations regarding accuracy, speed, power, and memory efficiency. We use four datasets, namely MNIST, CIFAR-10, ImageNet, and FaceNet, and four CNN models, namely LeNet, AlexNet, VGG-16, and MobileNet, to test our implementation. We compare our implementation with the existing FPGA-based CNN implementations, such as [4], [5], and [6], and show that our implementation achieves higher accuracy, faster speed, lower power, and lower memory consumption. We also show that our implementation can support different CNN models and datasets by reconfiguring the hardware architecture and applying different optimization techniques.

# Related Work

In this section, we review the related work on FPGA-based CNN implementations and highlight the differences and advantages of our work. We classify the related work into three categories, namely hardware architectures, optimization techniques, and software frameworks.

# Hardware Architectures

Several hardware architectures have been proposed for implementing CNNs on FPGAs, which can be broadly divided into two types: spatial architectures and temporal architectures. Spatial architectures map each neuron or operation to a dedicated logic block and perform them in parallel. Temporal architectures map multiple neurons or operations to a shared logic block and execute them sequentially. Spatial architectures can achieve high performance and parallelism but require more hardware resources and power consumption. Temporal architectures can achieve low hardware resources and power consumption but require more computation cycles and memory bandwidth. Therefore, spatial and temporal architectures have a trade-off between performance, power, and resource efficiency.

Some examples of spatial architectures are [4], [5], and [6], which implement CNNs on FPGAs using different types of logic blocks, such as DSPs, BRAMs, and LUTs. These architectures can achieve high performance and parallelism but consume more hardware resources and power. Some examples of temporal architectures are [7], [8], and [9], which implement CNNs on FPGAs using a single or a few logic blocks, such as DSPs or BRAMs. These architectures can achieve low hardware resources and power consumption but require more computation cycles and memory bandwidth.

Our hardware architecture is different from the existing spatial and temporal architectures, as it combines both types' advantages, achieving high performance, parallelism, power, and resource efficiency. Our hardware architecture consists of a reconfigurable array of processing elements (PEs) that perform convolution, pooling, and activation operations and a controller that manages the data flow and synchronization. Our hardware architecture can support different CNN models and datasets by reconfiguring the number and type of PEs and the interconnection network. Our hardware architecture can also reduce the computation, memory, and power consumption of CNNs by applying quantization, pruning, and compression techniques, which reduce the bit-width, number, and size of the network parameters and data. Therefore, our hardware architecture is more flexible, scalable, and efficient than the existing spatial and temporal architectures.

# Optimization Techniques

Several optimization techniques have been proposed to reduce the computation, memory, and power consumption of CNNs on FPGAs. These can be broadly divided into quantization, pruning, and compression. Quantization reduces the network parameters' bit-width and data, replacing the floating-point operations with fixed-point or binary operations. Pruning reduces the number of network parameters and data and removes the redundant or insignificant neurons, weights, or filters. Compression reduces the size of the network parameters and data and applies various encoding or compression schemes, such as Huffman coding, run-length encoding, or sparse matrix representation. These optimization techniques can reduce the computation, memory, and power consumption of CNNs but may also cause accuracy degradation or hardware overhead.

Some examples of quantization techniques are [10], [11], and [12], which implement CNNs on FPGAs using different bit-widths, such as 16-bit, 8-bit, or 1-bit. These techniques can reduce the computation, memory, and power consumption of CNNs but may also cause accuracy degradation or hardware overhead. Some examples of pruning techniques are [13], [14], and [15], which implement CNNs on FPGAs using different pruning methods, such as weight pruning, filter pruning, or channel pruning. These techniques can reduce the number of network parameters and data but may also cause accuracy degradation or hardware overhead. Some examples of compression techniques are [16], [17], and [18], which implement CNNs on FPGAs using different compression schemes, such as Huffman coding, run-length encoding, or sparse matrix representation. These techniques can reduce the network parameters and data size but may also cause accuracy degradation or hardware overhead.

Our implementation applies various optimization techniques, such as quantization, pruning, and compression, to reduce the computation, memory, and power consumption of CNNs on FPGAs without causing significant accuracy degradation or hardware overhead. Our implementation can select the optimal bit-width, pruning method, and compression scheme for each CNN layer and dataset according to the accuracy, performance, power, and memory requirements. Our implementation can also adapt to different CNN models and datasets by reconfiguring the hardware architecture and applying different optimization techniques. Therefore, our implementation is more accurate, efficient, and adaptive than the existing optimization techniques.

# Software Frameworks

Several software frameworks have been proposed to facilitate the implementation of CNNs on FPGAs, which can be broadly divided into two types: high-level frameworks and low-level frameworks. High-level frameworks provide a user-friendly interface and a high-level programming language, such as C, Python, or TensorFlow, for describing the CNN model and the FPGA implementation. Low-level frameworks provide a low-level interface and programming language, such as Verilog, VHDL, or OpenCL, for describing the CNN model and the FPGA implementation. High-level frameworks can simplify the design process and reduce the development time but may also cause performance degradation or resource inefficiency. Low-level frameworks can achieve high performance and resource efficiency but may also cause design complexity and increase the development time. Therefore, a trade-off exists between usability, performance, and resource efficiency for high-level and low-level frameworks.

Some examples of high-level frameworks are [19], [20], and [21], which provide a user-friendly interface and a high-level programming language, such as C, Python, or TensorFlow, for describing the CNN model and the FPGA implementation. These frameworks can simplify the design process and reduce the development time, but may also cause performance degradation or resource inefficiency. Some examples of low-level frameworks are [22], [23], and [24], which provide a low-level interface and a low-level programming language, such as Verilog, VHDL, or OpenCL, for describing the CNN model and the FPGA implementation. These frameworks can achieve high performance and resource efficiency but may also cause design complexity and increase the development time.

Our software framework is different from the existing high-level and low-level frameworks, as it combines both types' advantages, achieving high usability, performance, and resource efficiency. Our software framework takes a CNN model as input and performs various steps, such as network analysis, hardware synthesis, software compilation, and FPGA programming. Our software framework can analyze the network structure and parameters and select the optimal hardware configuration and optimization techniques for the target FPGA device and embedded system. Our software framework can also synthesize the hardware architecture, compile the embedded software, and program the FPGA device without the user's manual intervention or hardware design expertise. Therefore, our software framework is more user-friendly, fast, and effective than the existing high-level and low-level frameworks.

# Conclusion

In this paper, we proposed a novel implementation of CNN on an FPGA-based embedded system, which achieved high accuracy, performance, power, and memory efficiency. We designed a custom hardware architecture that exploited the parallelism and sparsity of CNNs and implemented various optimization techniques, such as quantization, pruning, and compression. We also developed a software framework that automatically generated the FPGA configuration and the embedded software from a given CNN model. We evaluated our implementation on several benchmark datasets and CNN models. We showed that it outperformed the state-of-the-art FPGA-based CNN implementations regarding accuracy, speed, power, and memory efficiency. We also showed that our implementation could support different CNN models and datasets by reconfiguring the hardware architecture and applying different optimization techniques. Our future work includes extending our implementation to support more CNN models and datasets and exploring more optimization techniques and hardware architectures for implementing CNNs on FPGAs.