Literature Review of CNN on FPGA-Based Embedded Systems

A Survey of Recent Advances and Future Directions

# Abstract

This document provides a literature review of Convolutional Neural Networks (CNNs) on FPGA-based embedded systems. It discusses the background and motivation for using CNNs and FPGAs for embedded systems, highlighting the advantages and challenges of both technologies. The document also discusses the critical aspects of implementing CNNs on FPGAs, including model optimization, architecture optimization, and system integration. Finally, the document highlights open issues and future directions for this research area, including model-architecture co-optimization and dynamic and adaptive systems.

KEYWORDS - Convolutional Neural Networks (CNNs), FPGA-based embedded systems, Model optimization, Architecture optimization, System integration, Compression, Pruning, Quantization, Sparsity, Parallelism, Pipelining, Tiling, Model-Architecture Co-Optimization, Dynamic and Adaptive Systems, Emerging Trends and Opportunities

# Introduction

Convolutional neural networks (CNNs) are a powerful class of deep learning models that have achieved remarkable results in various domains, such as computer vision, natural language processing, and speech recognition. CNNs are composed of multiple layers of neurons that perform convolution operations on the input data, followed by non-linear activation functions and pooling operations. CNNs can learn hierarchical and invariant features from large-scale data and can generalize well to new and unseen data. However, CNNs also have high computational and memory requirements, which pose significant challenges for their deployment on resource-constrained embedded systems, such as mobile devices, drones, robots, and wearable sensors. Embedded systems have limited power, memory, and bandwidth and often must operate in real-time and under dynamic environments. Therefore, designing efficient and robust CNN architectures and implementations for embedded systems is an active and important research topic.

Field-programmable gate arrays (FPGAs) are reconfigurable hardware devices that can implement custom logic circuits using programmable logic blocks and interconnects. FPGAs have several advantages over traditional processors and graphics processing units (GPUs) for implementing CNNs on embedded systems. First, FPGAs can exploit the parallelism and pipelining of CNNs at a fine-grained level and achieve high performance and energy efficiency. Second, FPGAs can adapt to the diverse and evolving CNN models and applications, providing flexibility and scalability. Third, FPGAs can support heterogeneous and hybrid computing and integrate with other hardware components, such as sensors, cameras, and memory modules. Therefore, FPGAs have emerged as a promising platform for accelerating CNNs on embedded systems.

In this document, we review the recent literature on CNNs on FPGA-based embedded systems and provide a comprehensive survey of the state-of-the-art methods, techniques, and challenges. We first introduce the background and motivation for using CNNs and FPGAs for embedded systems and present the primary design goals and trade-offs. We then discuss the critical aspects of implementing CNNs on FPGAs, including CNN model optimization, FPGA architecture optimization, and system integration. We also highlight some open issues and future directions for this research area. Finally, we conclude with a summary and some remarks.

# Background and Motivation

This section provides some background and motivation for using CNNs and FPGAs for embedded systems. We first briefly describe the basic concepts and components of CNNs and explain why they are suitable for embedded applications. We then introduce the main features and challenges of FPGAs and show how they can benefit from CNN acceleration.

* CNNs for Embedded Applications

CNNs are a type of artificial neural network that is inspired by the biological vision system. They consist of multiple layers of neurons that process the input data hierarchically and spatially. The input data can be images, videos, audio, text, or multi-dimensional signals. The output of each layer is a feature map, which represents the activation of the neurons in that layer. The feature maps are then fed to the next layer until the final output layer, which produces the desired result, such as classification, detection, segmentation, or generation.

The most common types of layers in CNNs are convolutional layers, pooling layers, and fully connected layers. Convolutional layers perform convolution operations on the input feature maps using a set of learnable filters or kernels. Each filter extracts a specific feature from the input and produces an output feature map. Pooling layers perform down-sampling operations on the input feature maps using a predefined function, such as max, average, or min. Pooling layers reduce the spatial size and complexity of the feature maps and enhance the robustness and invariance of the features. Fully connected layers perform linear transformations on the input feature maps using a matrix of weights and a bias vector. Fully connected layers act as classifiers or regressors and produce the final output of the CNN.

CNNs have several advantages for embedded applications. First, CNNs can learn complex and high-level features from large-scale data and achieve superior accuracy and performance compared to traditional methods. Second, CNNs can handle various types of data and tasks and support multimodal and cross-domain learning. Third, CNNs can be trained offline on powerful servers or cloud platforms and deployed on embedded devices with minimal or no retraining. Fourth, CNNs can be compressed and pruned to reduce their size and complexity and adapt to the resource constraints of embedded systems.

* FPGAs for CNN Acceleration
* FPGAs are hardware devices that can be programmed to implement any logic function using a large array of logic blocks and interconnects. Logic blocks are the basic computation units in FPGAs and can perform simple operations, such as arithmetic, logic, or memory. Interconnects are the wires that connect the logic blocks and can be configured to form different paths and circuits. FPGAs can be programmed using hardware description languages (HDLs), such as Verilog or VHDL, or high-level synthesis (HLS) tools like C or C++.
* FPGAs have several advantages for CNN acceleration. First, FPGAs can exploit the parallelism and pipelining of CNNs at a fine-grained level and achieve high performance and energy efficiency. FPGAs can implement multiple filters, feature maps, and layers in parallel and use dedicated hardware resources for each operation. FPGAs can also pipeline the execution of different stages of CNNs and overlap the computation and communication. Second, FPGAs can adapt to the diverse and evolving CNN models and applications, providing flexibility and scalability. FPGAs can reconfigure their logic blocks and interconnects to match the structure and parameters of different CNNs and support dynamic and partial reconfiguration. FPGAs can also scale their resources and performance up or down according to the workload and environment. Third, FPGAs can support heterogeneous and hybrid computing and integrate with other hardware components, such as sensors, cameras, and memory modules. FPGAs can communicate and cooperate with different processors and GPUs, forming a heterogeneous system-on-chip (SoC) or hybrid system. FPGAs can also interface with various sensors and cameras and perform pre-processing and post-processing on the data.

# CNN Implementation on FPGAs

This section discusses the key aspects of implementing CNNs on FPGAs, including CNN model optimization, FPGA architecture optimization, and system integration. We review the main methods and techniques proposed in the literature and compare their advantages and disadvantages. We also identify some of the challenges and limitations of the current approaches and suggest possible solutions and improvements.

* CNN Model Optimization

CNN model optimization aims to reduce the size and complexity of CNNs and improve their efficiency and robustness for embedded systems. CNN model optimization can be performed at different levels, such as network, layer, filter, or weight. CNN model optimization can be applied during the training phase, the inference phase, or both. CNN model optimization can use various techniques, such as compression, pruning, quantization, or sparsity.

Compression is a technique that reduces the number of parameters or operations of a CNN by applying a transformation or a coding scheme. Compression can be lossless or lossy, depending on whether the original information can be fully recovered. Compression can use different methods, such as matrix factorization, low-rank approximation, dictionary learning, or Huffman coding. Compression can reduce the memory footprint and bandwidth consumption of CNNs and improve their energy efficiency. However, compression can also introduce some accuracy loss or computation overhead and require additional hardware resources for encoding and decoding.

Pruning is a technique that removes the redundant or insignificant parameters or operations of a CNN by applying a threshold or a criterion. Pruning can be structured or unstructured, depending on whether the parameters or operations are removed regularly or irregularly. Pruning can use different methods, such as weight magnitude, weight sensitivity, filter similarity, or filter correlation. Pruning can reduce the computational complexity and latency of CNNs and improve their performance and power efficiency. However, pruning can also introduce some accuracy loss or sparsity and require additional hardware resources for indexing and masking.

Quantization is a technique that reduces the bit-width or the precision of the parameters or operations of a CNN by applying a rounding or a scaling scheme. Quantization can be uniform or non-uniform, depending on whether the parameters or operations are quantized with the same or different intervals. Quantization can use methods like fixed-point, floating-point, binary, or ternary. Quantization can reduce the arithmetic complexity and hardware cost of CNNs and improve their speed and energy efficiency. However, quantization can also introduce some accuracy loss or quantization error, requiring additional hardware resources for conversion and calibration.

Sparsity is a technique that exploits the zero or near-zero values of the parameters or operations of a CNN by applying a masking or a skipping scheme. Sparsity can be induced or inherent, depending on whether an external or internal factor makes the parameters or operations sparse. Sparsity can use different methods, such as thresholding, pruning, or regularization. Sparsity can reduce the memory access and computation cost of CNNs and improve their throughput and energy efficiency. However, sparsity can also introduce some accuracy loss or irregularity and require additional hardware resources for storage and execution.

* FPGA Architecture Optimization

FPGA architecture optimization aims to improve the utilization and performance of FPGA resources and match the characteristics and requirements of CNNs. FPGA architecture optimization can be performed at different levels, such as system level, module level, or component level. FPGA architecture optimization can be applied during the design phase, the synthesis phase, or both. FPGA architecture optimization can use various techniques like parallelism, pipelining, or tiling.

Parallelism is a technique that increases the concurrency and throughput of FPGA resources by performing multiple operations or tasks simultaneously. Parallelism can be spatial or temporal, depending on whether the operations or tasks are executed on different or the same hardware units. Parallelism can use loop unrolling, task partitioning, or data distribution. Parallelism can improve the performance and efficiency of FPGA resources and exploit the parallelism and diversity of CNNs. However, parallelism can also introduce resource overhead or communication costs and require additional hardware resources for synchronization and coordination.

Pipelining is a technique that increases the frequency and latency of FPGA resources by dividing an operation or a task into multiple stages or steps. Pipelining can be fine-grained or coarse-grained, depending on whether the stages or steps are small or large. Pipelining can use different methods, such as loop pipelining, task pipelining, or data pipelining. Pipelining can improve the performance and efficiency of FPGA resources and exploit the pipelining and locality of CNNs. However, pipelining can also introduce resource overhead or pipeline hazards and require additional hardware resources for buffering and control.

Tiling is a technique that increases the reuse and bandwidth of FPGA resources by dividing the data or the computation into multiple blocks or tiles. Tiling can be static or dynamic, depending on whether the blocks or tiles are fixed or variable. Tiling can use different methods, such as loop tiling, task tiling, or data tiling. Tiling can improve the performance and efficiency of FPGA resources and exploit the tiling and regularity of CNNs. However, tiling can also introduce some resource overhead or tiling overhead, and additional hardware resources are required for mapping and scheduling.

* System Integration
* System integration aims to combine the CNN model and the FPGA architecture into a complete and functional system and ensure the correctness and quality of the system. System integration can be performed at different levels, such as hardware, software, or application levels. System integration can be applied during the deployment, testing, or both. System integration can use various techniques like interface, communication, or verification.
* Interface is a technique that connects the CNN model and the FPGA architecture and enables the data and control transfer between them. Interface can be internal or external, depending on whether the connection is within or outside the FPGA device. Interfaces can use different methods, such as memory, peripheral, and network interfaces. The interface can ensure the compatibility and interoperability of the CNN model and the FPGA architecture, as well as support the data and control flow of the system. However, the interface can also introduce some resource overhead or interface overhead and require additional hardware resources for interfacing and configuration.
* Communication is a technique that transfers the data and control signals between the CNN model and the FPGA architecture and coordinates their execution and behavior. Communication can be serial or parallel, depending on whether the data and control signals are transferred one by one or in parallel. Communication can use different methods, such as bus, network, or direct. Communication can ensure the reliability and efficiency of the data and control transfer between the CNN model and the FPGA architecture, as well as support the communication and cooperation of the system. However, communication can also introduce some resource or communication overhead, and additional hardware resources are required for communication and arbitration.
* Verification is a technique that validates the functionality and performance of the system and detects and corrects the errors and faults of the system. Verification can be functional or non-functional, depending on whether the validation is based on the system's functionality or performance. Verification can use different methods, such as simulation, emulation, or prototyping. Verification can ensure the correctness and quality of the system and improve its robustness and reliability. However, verification can also introduce some resource overhead or verification overhead and require additional hardware resources for verification and debugging.

# Open Issues and Future Directions

In this section, we highlight some of the open issues and future directions for the research area of CNNs on FPGA-based embedded systems. We identify some of the current methods and techniques challenges and limitations and suggest possible solutions and improvements. We also discuss emerging trends and opportunities for this research area and propose new directions and perspectives.

* Model-Architecture Co-Optimization

One of the main challenges for implementing CNNs on FPGAs is to optimize both the CNN model and the FPGA architecture and achieve the best trade-off between performance, accuracy, and resource consumption. However, most existing methods and techniques focus on either the CNN model optimization or the FPGA architecture optimization and do not consider the interaction and co-dependence between them. For example, some methods optimize the CNN model by applying compression, pruning, quantization, or sparsity but do not consider the impact of these techniques on the FPGA architecture, such as the memory layout, the data access, or the computation schedule. Similarly, some methods optimize the FPGA architecture by applying parallelism, pipelining, or tiling but do not consider the impact of these techniques on the CNN model, such as the accuracy loss, the quantization error, or the sparsity pattern. Therefore, there is a need for model-architecture co-optimization methods and techniques that can jointly optimize the CNN model and the FPGA architecture and exploit the synergy and complementarity between them.

One possible solution for model-architecture co-optimization is to use a holistic and unified framework to integrate the CNN model optimization and the FPGA architecture optimization and perform them simultaneously or iteratively. Such a framework can use a common objective function or metric to capture the trade-off between performance, accuracy, and resource consumption and guide the optimization process. Such a framework can also use a common representation or a language that can describe the CNN model and the FPGA architecture and enable communication and coordination between them. Such a framework can also use a standard tool or platform to implement the CNN model and the FPGA architecture and support their evaluation and verification.

* Dynamic and Adaptive Systems

Another challenge for implementing CNNs on FPGAs is to cope with embedded systems' dynamic and uncertain environments and adapt to the applications' changing and evolving requirements and conditions. However, most existing methods and techniques assume that the CNN model and the FPGA architecture are fixed and static and do not consider the possibility and necessity of updating and reconfiguring them. For example, some methods optimize the CNN model and the FPGA architecture for a specific application or task but do not consider the input data's variation, diversity, output result, or user preference. Similarly, some methods optimize the CNN model and the FPGA architecture for a specific environment or a scenario but do not consider the change or the uncertainty of the workload, the power, or the temperature. Therefore, there is a need for dynamic and adaptive methods and techniques that can update and reconfigure the CNN model and the FPGA architecture and exploit the reconfigurability and flexibility of FPGAs.

One possible solution for dynamic and adaptive systems is to use a feedback and learning mechanism to monitor and analyze the system's performance, accuracy, and resource consumption and adjust and improve the CNN model and the FPGA architecture accordingly. Such a mechanism can use a feedback loop or a controller that can collect and process the information and the signals from the system and provide guidance and commands to the system. Such a mechanism can also use a learning algorithm or policy to learn and optimize the CNN model and the FPGA architecture and support the system's adaptation and evolution.

* Emerging Trends and Opportunities

In addition to the challenges and limitations of the current methods and techniques, there are also some emerging trends and opportunities for the research area of CNNs on FPGA-based embedded systems. The rapid development and innovation of the CNN models, the FPGA devices, and the embedded applications drive these trends and opportunities. For example, some of the emerging trends and opportunities are:

* New and advanced CNN models, such as deep residual networks, generative adversarial networks, or attention-based networks, can achieve higher accuracy and performance and support more complex and diverse tasks, such as image synthesis, video analysis, or natural language understanding.
* New and improved FPGA devices, such as heterogeneous FPGAs, 3D FPGAs, or optical FPGAs, that can provide higher performance and efficiency and support more functionality and diversity, such as mixed-precision computing, memory stacking, or photonic interconnects.
* New and emerging embedded applications

# Conclusion

In conclusion, this literature review comprehensively surveys the state-of-the-art methods, techniques, and challenges in implementing CNNs on FPGA-based embedded systems. It discusses the background and motivation for using CNNs and FPGAs for embedded systems, highlighting the advantages and challenges of both technologies. The document also discusses the critical aspects of implementing CNNs on FPGAs, including model optimization, architecture optimization, and system integration. Finally, the document highlights open issues and future directions for this research area, including model-architecture co-optimization and dynamic and adaptive systems.