Design and Implementation of Scalable Accelerator for Semantic Segmentation in Self-driving

Xin Liu

Abstract— **The application of Convolutional Neural Networks (CNNs) has significantly accelerated the development of semantic segmentation, particularly in the domain of autonomous driving. Semantic segmentation is crucial for enabling autonomous vehicles to accurately perceive their surroundings and make real-time decisions. However, the increasing computational complexity has led to a substantial rise in power consumption, hindering the progress of self-driving technology. While Field Programmable Gate Arrays (FPGAs) offer a means to accelerate network inference, achieving an optimal balance between accuracy and speed remains a significant challenge. This paper investigates state-of-the-art semantic segmentation models and their corresponding optimization techniques. We summarize the critical requirements for system flexibility when mapping models to embedded FPGAs. Based on these requirements, we propose a reconfigurable semantic segmentation accelerator that integrates hardware optimization and data quantization strategies. The data quantization strategy reduces the bit width to 8 bits without any discernible loss in accuracy. To further reduce inference time, the network operators are implemented and optimized directly in hardware. Additionally, an instruction-controlled data flow is employed to support future updates and scalability. To enhance coding efficiency and reusability, we utilize SpinalHDL, an emerging hardware description language embedded in Scala, a high-level programming language, for the development of the proposed accelerator. The performance of the design is evaluated on the Virtex UltraScale+ VU9P FPGA platform, yielding accuracies of 74.3% on the CamVid dataset and 72.1% on the Cityscapes dataset, with a processing speed of 24 FPS, approaching real-time performance. This work paves the way for more energy-efficient and scalable solutions for autonomous driving systems, with potential for real-world deployment in various safety-critical environments.**

Index Terms—**FPGA, Semantic segmentation, Scalable design, Accelerator**

I. INTRODUCTION

The advent of Convolutional Neural Networks (CNNs) has initiated a revolutionary transformation in the field of computer vision. Compared to traditional methods, CNNs offer significant advantages. including the ability to automatically learn and extract image features, effectively handle large-scale image datasets, and enhance accuracy across various tasks[1]. Semantic segmentation, a vital component of CNNs, plays a crucial role in numerous industries, particularly in the realm of self-driving technology. By precisely classifying of each pixel captured by onboard cameras, semantic segmentation enables autonomous vehicles to achieve a comprehensive understanding of their surrounding environment, including roads, pedestrians, and other vehicles. This detailed perception is essential for the safe navigation and decision-making processes of autonomous vehicles. Specifically, self-driving cars must accurately segment cars, pedestrians, road signs, and other objects in real-time to make precise control decisions, ensuring safety and robustness in diverse driving conditions[2].

Nevertheless, in practical contexts, the pursuit of accuracy alone is insufficient. Improving accuracy often leads to increased computational complexity, which in turn results in a sharp decline in recognition speed. To enhance the processing speed of self-driving systems, the utilization of Graphics Processing Units (GPUs) has become a prevalent approach for network training and inference tasks. The parallel processing capabilities of GPUs enable low latency and real-time performance when handling large models[3]. Despite the high efficiency, the significant power consumption of GPUs presents a challenge for the advancement of self-driving technology. It is imperative that self-driving systems process substantial quantities of real-time data in a timely and efficient manner while simultaneously maintaining low power consumption. This is crucial for ensuring the safety and stability of the system.

In order to reduce power consumption, researchers have initiated an investigation into the potential of using embedded processors for the execution of self-driving tasks. Compared to GPUs, embedded processors exhibit markedly reduced power consumption, thereby rendering them more appropriate for real-time applications. Notable among these are field-programmable gate arrays (FPGAs) and application-specific integrated circuits (ASICs), which have attracted considerable interest from researchers. While ASICs could potentially achieve significant energy savings due to their custom-designed circuits, they lacked flexibility and had lengthy development cycles, which proved disadvantageous in the context of the rapidly evolving field of self-driving[4]. In contrast, FPGAs have the potential to markedly enhance the speed of inference by modifying a series of internal ASIC modules, including DSP blocks, block RAM, and the necessary interface cores[5]. Compared to software implementations on multi-core processors and GPUs, FPGAs achieved competitive energy efficiency (approximately 10-50 GOP/s/W) and low-latency inference , which is especially important for applications in the self-driving field that required low latency and long-term stable operation[7].

This paper presents a reconfigurable hardware accelerator designed to address the limitations of existing semantic segmentation solutions, particularly regarding the balance

Manuscript received March 10, 2025
Xin Liu, School of computer science and technology, Tiangong University, Tianjin, China

between accuracy and computational efficiency. We selected the BiSeNetV1[8] network, a lightweight architecture known for achieving a trade-off between model complexity and segmentation accuracy. To facilitate deployment on resource-constrained hardware platforms, we employed a quantization strategy that effectively reduces model size without compromising inference accuracy. This approach optimizes resource utilization, making it feasible for implementation on platforms with limited capacity.The proposed quantization method was rigorously validated across multiple datasets, demonstrating its effectiveness in maintaining high accuracy while minimizing resource consumption. To further enhance computational efficiency, we integrated several optimization techniques, including loop unrolling, tiling, and data swapping, which improve parallelism in convolution operations and streamline the overall computation process, thereby reducing processing time.DSP optimization was applied to minimize on-chip memory access frequency and reduce data transmission overhead, resulting in more efficient use of hardware resources — an essential consideration for devices with constrained capabilities. The design of other operators was tailored to hardware constraints, further accelerating network inference and achieving simultaneous optimization of both resources and processing speed.Additionally, the use of SpinalHDL, a high-level hardware description language, allowed for flexible configuration of FPGA parameters. This flexibility ensures that developers can adjust hardware implementations without focusing on low-level details, thereby enhancing adaptability across various hardware configurations. An instruction-based execution flow control was also implemented to support future network upgrades, significantly improving the flexibility and reconfigurability of the hardware design.The proposed accelerator was deployed on the Virtex UltraScale+ VU9P FPGA and evaluated on the CamVid and CityScapes datasets. It achieved an accuracy of 74.3% on CamVid and 72.7% on CityScapes, with processing speeds nearing real-time performance. These results demonstrate the accelerator's potential in real-time semantic segmentation tasks, offering both enhanced speed and resource efficiency compared to existing solutions.

This document is organized as follows. Section 2 introduces related work and highlights its shortcomings. Section 3 introduces the design workflow and the specific hardware architecture. It includes an analysis of the network structure and the implementation of specific hardware operators, along with proposed optimization strategies for the accelerator. Section 4 presents the performance results of the final solution, including experimental process, accelerator execution time, and comparisons with other FPGA-based works. Section 5 concludes the work and providing suggestions for future work.

II. Related Work

An autonomous driving system is comprised of four principal components:perception,self-localization, prediction, and decision-making. Among these, perception is of fundamental importance, as subsequent prediction and decision-making are contingent upon it. The process of perception, however, begins with the acquisition of relevant information from the vehicle's surroundings[9]. In recent years, there has been a notable increase in interest in object detection. This surge in interest has led to the proposal of various optimization schemes, aimed to enhance the performance of object detection[10]-[11]-[12]. Moreover, perception is not solely concerned with object detection; it also encompasses semantic segmentation. A principal distinction between semantic segmentation and object detection is that the former offers a more comprehensive understanding of the scene, thereby enabling more precise prediction and decision-making[13]. Consequently, the existing solution cannot be readily applied to optimize semantic segmentation.

Initial developments in semantic segmentation were initiated by Fully Convolutional Networks (FCNs), which expanded network models to facilitate pixel-level predictions, thereby pioneering end-to-end training[14]. In order to enhance the precision of their results, numerous semantic segmentation models started encoding more spatial information or expanding the receptive field[15]-[16]-[17], resulting in a significant increase in the overall model size. Networks such as U-Net and SegNet applied an encoder-decoder architecture to enhance segmentation accuracy by recovering spatial information[18]-[19]. However, some information remained difficult to recover. which greatly enhance the capture of multi-scale contextual information and significantly improve segmentation performance[15]-[20]-[21]. Despite these advancements in accuracy, these methods also increased model complexity and computational cost, rendering real-time applications challenging.The BiSeNetV1 network adopts a bilateral network structure to cope with the loss of spatial information and the shrinkage of receptive fields, while preserving spatial details as much as possible and reducing computational costs, using fewer convolutional layers. There has been a certain improvement in real-time performance[8].

Previous research sought to accelerate inference by reducing the size of the input image[22] or by pruning network channels[23]. Nevertheless, this approach frequently resulted in a reduction in accuracy, which was inadequate to fulfill the criteria for precision. To address these issues, a pipelined structure using depthwise separable convolutions was proposed[24], which decomposes standard convolutions into depthwise convolutions (for filtering) and pointwise convolutions (for combining features), resulting in the significant reduction of the computational load by approximately nine times. However, this approach might lead to the isolation of information between channels, thereby impairing the network's ability to capture inter-channel correlations and affecting overall performance. Consequently, focusing solely on the network was insufficient to achieve an optimal balance between speed and accuracy. This led researchers to investigate the potential of embedded processors.

Nevertheless, inexpensive embedded processors are capable of achieving a processing speed of tens of GFLOPs (one billion floating-point operations) per second, which is insufficient for near real-time processing of semantic segmentation. If speed is pursued without consideration of

other factors, dedicated design circuits can achieve optimal performance and energy efficiency. For example, Li described a dedicated CMOS image sensor (CIS) chip that achieves optimal computational efficiency and power consumption through the use of specialized circuit design[25]. However, the configurability of customized circuits is severely limited, which presents a significant challenge in subsequent algorithm optimization. FPGA serves as a compromise solution, offering the potential for specialized hardware accelerators for various neural network models. Its internal hardware circuits are fixed, while its reconfigurability is superior to that of ASICs. High performance efficiency in network inference is demonstrated by FPGA primarily due to its ability to be reconfigured optimally for different network models.

A number of hardware accelerators for semantic segmentation have already been put forth for consideration. For example, Ghielmetti introduced a hardware acceleration scheme based on ENet, implemented using HLS4ml[3]. This scheme was converted to hardware code through high-level synthesis (HLS) and reduced resource utilization through quantization and filter pruning, achieving an image processing latency of 4.9 ms. However, only 36.8% of the mean Intersection over Union (mIoU) was achieved in the dataset. Despite the advantages of HLS in providing a more abstract representation of external modules and interfaces, thereby facilitating the implementation of intricate control logic, it is currently deficient in the capacity to pursue the optimization potential inherent to RTL design at a more granular level. To further enhance precision, Mori proposed the implementation of the DeepLabv3+ network structure, which has demonstrated considerable advancements in algorithmic accuracy[26]. In an effort to reduce inference time, genetic pruning of channels was employed to minimize the model's parameter count. Nevertheless, data transfer optimization was insufficient, resulting in significant delays. Furthermore, the network itself was highly intricate, with an ultimate inference time of 0.67s, rendering it unsuitable for real-time applications.

In order to reduce the complexity of the network and facilitate its deployment to hardware accelerators, a number of optimization techniques have been proposed. A common method is to employ quantization techniques to reduce the size of the data model by utilizing more straightforward weight representations[27]. Quantitative techniques reduce the bit width representation of parameters from 32-bit floating-point to lower bit widths, thereby markedly reducing the area and power consumption of model inference[28]. Moreover, ternary weight networks[29] and binary neural networks employ quantization of weights to lower bit widths (such as ternary and binary), thereby reducing storage requirements and computational complexity. It is important to note, however, that extreme quantization can have a significant impact on the accuracy of the model. For instance, while binarized networks, while optimal in data size, require 2 to 11 times more operations and weights than networks with 8-bit fixed-point weights to achieve similar accuracy on small networks[30]. Analysis in Gysel demonstrated that 8-bit fixed-point data is almost as accurate as 32-bit floating-point data[31].

An additional approach to model optimization is model compression, which entails the reduction of the number of weights or activations to lower memory and computational requirements. For instance, Han applied Huffman coding to trim and compress model data, predominantly in fully connected layers, resulting in a 91% reduction in the number of weights without compromising accuracy[32]. However, weight trimming was less effective in convolutional layers, where activation functions were more commonly employed to set outputs to zero rather than trimming weights, thereby reducing computation. Given that semantic segmentation networks lack fully connected layers, this technique is more suited to object detection.

A multitude of hardware accelerators have endeavored to achieve a balance between efficiency and flexibility through the implementation of diverse optimization techniques. However, the rapid evolution of algorithms presents a significant challenge for hardware architects. This difficulty increases the gap between algorithms and hardware accelerators[33]. To address this bottleneck, a synchronous dataflow architecture based on hardware reconfiguration technology was proposed by[34]. The FPGA is reconfigured, allowing the architecture and hardware resources to be adjusted according to different network layers. This adjustment achieves significant acceleration. However, a drawback exists: the accelerator needs to be reconfigured for different sub-graphs.

This paper proposes a novel configurable semantic segmentation accelerator using SpinalHDL, which is more user-friendly compared to traditional methods. SpinalHDL simplifies RTL code generation, making it easier for researchers to understand and apply these codes while providing greater flexibility in hardware configuration. Concurrently, the hardware implementation of the operators within the network has been completed, and through network analysis, optimization operations have been performed on convolutions with excessive computational complexity, thereby conserving resources and enhancing computational efficiency. In particular, a lightweight instruction set was devised. The controller parses instructions to control the data flow, ensuring that the accelerator is not limited to supporting specific network architectures. In future development, the corresponding data flow configuration can be rapidly modified to align with different algorithmic models. This modification markedly enhances the efficiency of the development process, rendering it well-suited to a multitude of application scenarios.

III. The Proposed Method

A. Network Model Analysis

BiSeNetV1 is designed to balance accuracy and processing efficiency in semantic segmentation by utilizing a dual-branch structure. The Spatial Path focuses on extracting high-resolution features, preserving image details and maintaining real-time performance through shallow convolutional layers. In contrast, the Context Path captures global semantic features, which are aggregated using deep convolutional layers and global average pooling[8]. The network operates in five stages. In the first stage, spatial details are extracted using convolutional layers. The second stage reduces the feature map size via max pooling, which

increases the receptive field. In the third stage, rich contextual information is captured during downsampling, facilitated by the ResNet18 backbone. The fourth stage integrates spatial and semantic information through the Feature Fusion Module. Finally, the fifth stage restores the resolution of the segmentation map through upsampling.An analysis of the latency and floating-point operations (FLOPs) distribution of various operators in the BiSeNetV1 model (as shown in Fig.1 revealed that most computational resources are allocated to the stages responsible for spatial feature extraction and deep semantic information capture. Convolution operations dominate in terms of both latency and FLOPs, indicating their significant impact on overall computation. Convolution primarily involves multiply-accumulate operations, and while these operations are computationally straightforward, their volume makes sequential execution on a CPU time-consuming. Thus, parallel computation on hardware provides an effective optimization strategy, with the primary focus on accelerating convolution operators.In addition to convolution, operators such as addition (add), multiplication (mul), and mean pooling also contribute to the computational load. Although their frequency is lower, these operations can still be optimized through hardware acceleration to improve overall performance. To further conserve hardware resources, the network backbone is trained using ResNet18, and the network model is modified by adjusting the activation function to Leaky ReLU. Large convolution kernels are uniformly replaced with standard 3x3 convolution kernels. After training, the network achieved accuracies of 75.4% on the CamVid dataset and 73.6% on the CityScapes dataset. Detailed optimization strategies and techniques will be discussed further in subsequent sections.

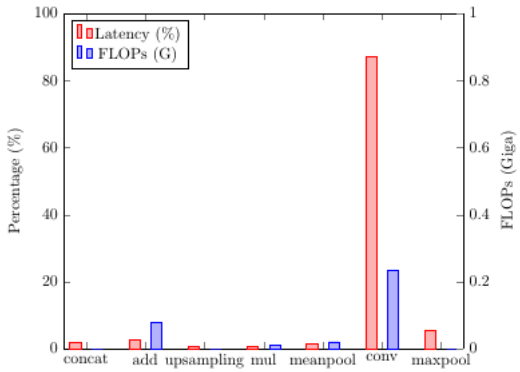

Fig.1 Latency and FLOPs Distribution of Operations

B. *Hardware Implementation Workflow Overview*

The proposed hardware accelerator workflow is illustrated in Fig.2. The workflow is comprised of five stages which integrate the design of the hardware accelerator with the corresponding instructions. The initial stage of the process entails training the model using the PyTorch framework. This involves training the dataset, optimizing the network hyperparameters, and quantizing the model to reduce the number of MAC operations, thereby facilitating the deployment of the model on hardware with greater ease. The second stage entails the description of the hardware architecture corresponding to the network model, which is achieved through the use of SpinalHDL. SpinalHDL's highly parameterized nature facilitates rapid automated hardware design, including hardware implementation and parameter configuration. The third stage involves using Vivado IDE to simulate, synthesize, and implement the RTL code and TCL scripts generated by SpinalHDL, producing the necessary hardware bitstream. The fourth stage involves transferring the instructions, model data, and weight information from the PC to the FPGA device. This stage ensures the accurate transmission of data and the establishment of an appropriate interface for communication with the FPGA. Ultimately, the network application runs on the FPGA in the fifth stage.

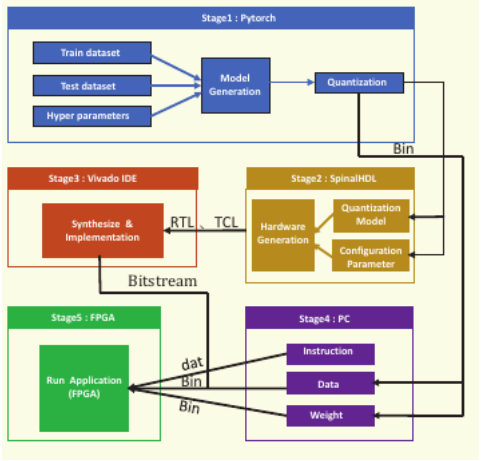

Fig.2 Workflow overview

C. *The Proposed Hardware Architecture*

We adopted a collaborative approach where the PC and FPGA worked in close conjunction to execute data-intensive tasks, as illustrated in Fig.3. The initialization process, which entailed the loading of input feature maps, weight vectors, and operation instructions, was conducted by the PC, while the FPGA was responsible for the efficient processing of data. Control signals and instructions were transmitted from the PC to the FPGA via the AXI-Lite bus and stored in the instruction register. Moreover, the transfer of computational data and results were efficiently transferred between the PC and FPGA's DRAM was conducted with optimal efficiency via the high-throughput AXI-Stream bus. Upon accelerator initialization, the instructions stored within the register were parsed by the controller, which also scheduled computational resources and managed data flow between DRAM and on-chip buffers. A DRAM interface module was designed to optimize data flow and memory access performance. To enhance isolation and efficiency, the on-chip buffers were partitioned into weight, output, and data buffers. The computation module was subdivided into two distinct sub-modules, namely the Conv and Shape sub-modules. The Conv module was responsible for performing convolution operations, while the shape module included units such as maximum pooling, concatenation (concat), add, upsampling, mul, and mean pooling, which were utilized to support diverse operations. The ports of the multiplexer and demultiplexer, as well as the arbitration module, were described using SpinalHDL, facilitating parameterization and configuration to accommodate different numbers of functional modules and markedly enhancing iterative development efficiency. When data flowed to the Shape module, the demultiplexer module orderly distributed tasks, and after operations were completed, the results were collected by the multiplexer module and transmitted to DRAM in accordance with the instructions.

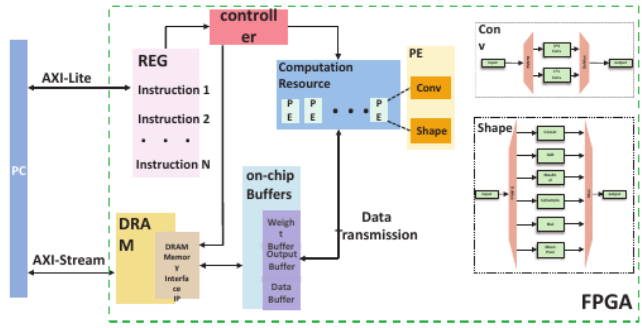

Fig.3 Accelerator process

CONV As indicated in Fig.1, convolution is the most resource-intensive and time-consuming computation in the network. During the convolution process, the energy consumed by data movement and memory access was found to be considerably higher than that of the convolution computation itself[35]-[36]. Due to the limitations of bandwidth, the cost of transferring data from external memory to internal memory was considerably higher than the transfer cost between on-chip memories[28]. The convolution operation primarily comprises a series of multiplication and accumulation cycles,which are performed as the window slides. To enhance efficiency, we opt to perform unfolding parallel computation on specific layers of the convolutional loop, encompassing the kernel size, input channel, and output channel. This approach leads to a notable reduction in computation time.This approach enhances computational efficiency while avoiding an unacceptable increase in hardware resources and power consumption.

We propose an efficient convolutional module design, capable of flexibly handling 3x3 and 1x1 convolution operations. The design employs loop optimization techniques, including loop unrolling, loop tiling, and loop permutation [29], to enhance computational efficiency and reduce energy consumption associated with data movement and memory access. As illustrated in Fig.4, prior to entering the convolutional module, the feature maps are rearranged and concatenated into a 3x3 convolution form through a multiplexer to guarantee data flow consistency and module reusability.

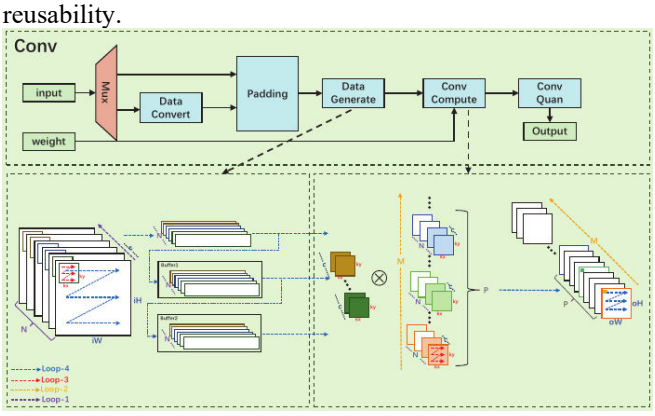

Fig.4 Implementation of convolution

The padding requirements are determined by the padding module based on instructions, thus avoiding performance degradation. A double-buffering strategy is employed by the data generation module to cache two rows of input feature map data, thereby reducing external memory access and accelerating the computation process. The convolution computation order is adjusted by adopting a channel-first strategy, extracting data blocks of size $k_x \times k_x \times C$ from local input feature maps and simultaneously computing the $k_x \times k_x$ points of the C convolution kernels are simultaneously computed. This approach achieved parallelism in terms of input channels, output channels, and convolution kernel size.

Following the completion of each computation, the next set of C channel data is calculated using the current convolution kernel's subsequent set of C channel data. The data is accumulated in the temporary buffer until all channels of the point have been computed, resulting in a complete output feature map point for the C channels. Subsequently, the input feature map is traversed from the initial channel of the aforementioned point. The calculations are performed with the C channels of the next set of P convolution kernels. This process continued until all convolution kernels and all channels of the local point in the input feature map have been computed. The sliding window is then moved along the row direction to the next pixel, continuing until the entire row is processed. It then moved down to the next row, repeating this process until the entire output feature map had been computed.

In the design of our accelerator, the input channel N is divided into N/C groups, with each group containing C channels. This transformation results in the conversion of the original single-layer loop into a two-layer loop structure. This allows acceleration of the C iterations of the inner loop by leveraging hardware parallelism, resulting in a speedup of N/C compared to the original serial computation. Furthermore, it enables an input channel parallelism of C. During each computation, the C channels of the input feature map are processed with the corresponding C channels of the convolution kernel, resulting in intermediate accumulation for one output channel, which is temporarily stored in a buffer. Once the computation and accumulation for a single C-channel block are complete, N/C iterations are necessary to compute all channels of the input feature map, thereby obtaining the complete output for a single channel. Following the completion of the convolution, quantization is required. However, if the output channels are not computed in parallel, each quantization would process only a single point from a given output channel of the feature map, thereby underutilizing the FPGA's parallelism.

To address this issue, the output channels are grouped. The M convolution kernels are divided into M/P groups, and the kernels within each group are computed simultaneously, achieving an output channel parallelism of P. The results of each group are stored in a temporary buffer and are accumulated with the results from the next group of P convolution kernels. This requires M/P iterations to compute the partial sums for all output feature map channels for one point.

Furthermore, convolution kernels are typically of a relatively small size, and larger kernels could be replaced by multiple smaller ones. Therefore, the $k_x \times k_x$ window of the input feature map and the corresponding convolution kernel rows and columns are fully unrolled, allowing the computation of $k_x \times k_x$ pixels in parallel within a single clock cycle. The $k_x \times k_x$ sized input feature map and kernel weights are transmitted in conjunction to the corresponding DSP units, where their computations are synchronized within a single clock cycle. This is achieved through the use of an

adder tree, which accumulates the partial results and temporarily stores them in a buffer.

The input feature map is reused N/C times, and the weights were reused $O_h \times O_w$ times, effectively reducing memory access costs. The convolution quantization module maps the output, ensuring efficient representation and transmission within the constraints of the FPGA resource constraints, and producing the final quantized convolution result.

Maxpool The max pooling layer is employed as a downsampling technique, whereby the spatial dimensions of the feature maps are reduced by selecting the maximum value within the pooling window. Typically, a 2x2 pooling window is utilized, which not only minimizes the computational load of subsequent layers but also enhances the model's resistance to spatial transformations and distortions, thereby preserving critical features such as edges and textures[37]. The data processing in our max pooling implementation, depicted in Fig.5 is carried out based on row parity using a Mux. Odd rows are initially stored in the col buffer, and upon the arrival of even rows, they are element-wise compared with the data in the col buffer, with the maximum values then stored in the row buffer. This approach optimizes both data flow and processing speed, thereby reducing the frequency of memory accesses.

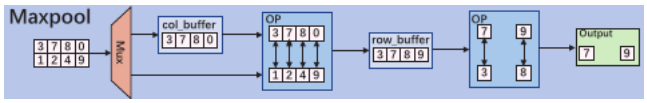

Fig.5 Implementation of MaxPool

Concat, Add The process of feature integration typically entails the concatenation and addition of data. The concat operation merges input tensors along a specified dimension, thereby maintaining the independence of each feature. This enables the network to learn diverse features. The add operation performs an element-wise addition of two tensors along the same dimension, effectively fusing the features. This operation is commonly employed in residual networks to mitigate issues related to gradient vanishing. We proposes an efficient hardware implementation method for integrating feature maps through Concat and Add operations. As illustrated in Fig.6, this method employs flexible instruction control to enable dynamic switching between concat and add operations, thereby optimizing hardware resource utilization and enhancing processing efficiency. Initially, the input data is stored in buffers to ensure its proper formatting prior to quantization. The quantization operation converts input data from high-precision format to optimized fixed-point representation, thereby reducing the computational load and adapting to hardware constraints, and minimizing precision loss. Based on instructions specific to neural network layers, the hardware unit selects between the add or concat operations. Add involves the element-wise addition of data from disparate buffers, whereas concat integrates data from two buffers along a specified dimension, thereby facilitating enhanced information integration. Subsequent to the execution of these operations, the processed data is forwarded to the subsequent network layer or utilized for further analysis, thereby achieving effective feature map integration and downsampling.

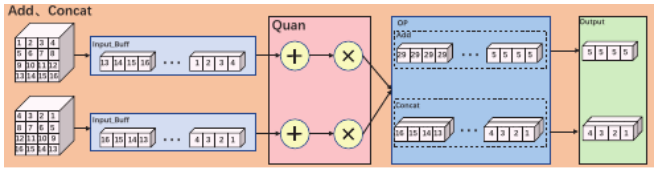

Fig.6 Implementation of Concat, Add

Upsampling In the process of extracting network features, the dimensions of the input feature map frequently undergo a gradual reduction. It is common practice to employ upsampling techniques in order to restore the feature map to its original dimensions. Among these techniques, interpolation methods enlarge the feature map by inserting new pixels between existing ones. The most prevalent methods include nearest neighbor and bilinear interpolation. Nearest neighbor interpolation is preferred method for real-time processing tasks due to its rapid computation time. As illustrated in Fig.7, the input feature map is initially stored in the input buffer for the purpose of temporarily holding the current feature map data. The operational unit executes a copy operation on each buffer element, duplicating resulting in the replication of each input element into multiple output elements aligned in rows or columns. This procedure directly generates an enlarged feature map and temporarily stores the duplicated data elements in the row buffer. As new input data entered the input buffer, the row buffer data is output synchronously within the same clock cycle, facilitating pipeline operation and enhancing data processing efficiency. This pipeline strategy ensures processing continuity, maximizes throughput, significantly reduces time costs, and enhances FPGA performance in upsampling tasks.

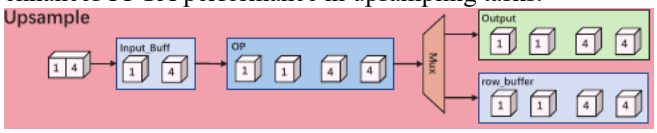

Fig.7 Implementation of Upsampling

Mul Due to differing levels of feature representation between the two types of features in the network, direct merging is impractical. The mul operator plays a critical role in the reweighting of features through the utilisation of weight vectors. This process enables the dynamic adjustment of the spatial feature map within each channel, with varying weights assigned to different spatial regions. The objective is to preserve and enhance high-weight features while attenuating low-weight ones. This adaptive weighting strategy enables the network to focus on regions rich in information content, thereby enhancing sensitivity to crucial features and refining boundary delineation precision in segmentation tasks[8]. As illustrated in Fig.8, this module receives inputs from two distinct buffers: one containing activated weights and the other comprising the input feature map. These weights serve as the primary filters for feature extraction, remaining static throughout the computations. The input feature map data streams continuously. Within the operational unit, each element of the input feature map undergoes element-wise multiplication with its corresponding weight value. During each processing cycle, the operational unit multiplies the weights from the weight buffer with corresponding feature map data from the input buffer, accumulating the results to derive a single output feature value. These accumulated results are subsequently quantized and directly forwarded to the next network layer.

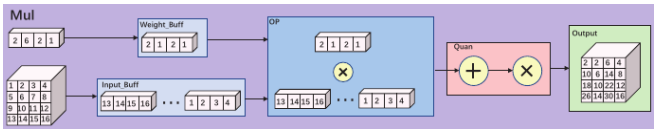

Fig.8 Implementation of Mul

Meanpool In order to guarantee the greatest possible receptive field and to incorporate global contextual information into the feature map, global average pooling is employed, thus enabling the network to capture comprehensive semantic information. This process involves the condensation of spatial features into a single vector, while ensuring the preservation of the channel dimension. Subsequently, the vector is subjected to a 1x1 convolution and batch normalization, resulting in the generation of a channel attention map. This approach not only reduces the model's complexity but also enhances the network's ability to learn essential features by globally aggregating information, thereby improving semantic integration across different scales[8]. Illustrated in Fig.9, input feature map data first enters a multiplexer, then row-wise temporarily resides in the column buffer based on buffer availability. The column buffer temporarily holds single-row feature map data for subsequent accumulation. As each row is accumulated in the column buffer, new feature map data arriving is accumulated element-wise with existing data at corresponding positions in the buffer. This accumulation ensures merging of feature map data at identical positions, forming the basis for subsequent average value computation. Once all input data is processed, the accumulated values are divided by the total number of elements in the feature map within the operational unit in order to compute the global average value for each channel. These calculated averages undergo further processing via the quantization unit, which enables the data format to be adjusted. This results in a reduction in representation complexity and an optimization of overall computational efficiency and resource utilization. Finally, the processed data, which are now quantized averages, are output for use in subsequent layers of the deep neural network.

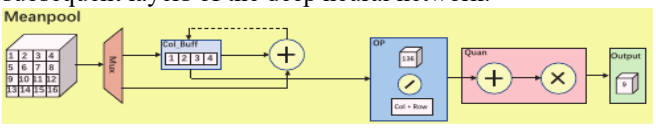

Fig.9 Implementation of Meanpool

Controller The controller employs 32-bit fixed-length instructions to oversee the data path of the accelerator. A specialized instruction set is defined for the purpose of executing neural networks. This includes control and status instructions for conv and shape operations, configuration parameters for operators, and direct memory access (DMA) control instructions, as shown in Table 1. Upon the commencement of an operator's execution, the initial instruction is first retrieved from the register by the controller, the operator type is determined, and the corresponding computational task is assigned to the appropriate processing unit. In order to guarantee the reconfigurability of the network, the instructions were responsible for controlling the configuration parameters of the computational units. By analyzing the configuration parameter instructions, different configurations could be applied to various operators, thereby markedly enhancing the efficiency of the development process. Furthermore, the controller is responsible for managing data transmission throughout the computation process. To enhance data transfer efficiency, the accelerator employs DMA control. Consequently, the DMA is informed by the controller of the data size and address space for read and write operations during each computation, ensuring optimal data processing. After completing the computation of each layer, the controller returns the status information of that layer. Upon receiving the correct return signal, the controller initiates the computation for the next layer.

Table1 Example of supported instructions

Instruction	Width	Section	R/W	Description
Conv/shape control	32	[3:0]	W	Execute specific operator
Conv/shape state	32	[3:0]	R	Read the operator state
	32	[31:22]	W	Channel_in
IMG_in shape	32	[21:11]	W	Col_in
	32	[10:0]	W	Row_in
	32	[31:22]	W	Channel_out
IMG_out shape	32	[21:11]	W	Col_out
	32	[10:0]	W	Row_out
Quan param	32	[31:0]	W	Scale、Zero
	32	[31]	W	En_Stride
	32	[30]	W	En_Padding
Conv Config	32	[29]	W	En_Activation
	32	[28:27]	W	Conv Type
	32	[26:0]	W	Weight Num
DMA Size	32	[31:0]	W	Write/Read Size
DMA Addr	32	[31:0]	W	Write/Read Addr

D. The proposed Optimization Method

Agile Development Technique In the contemporary field of hardware design, the enhancement of design efficiency has emerged as a prominent area of focus, alongside the optimization of performance. The popularity of agile development techniques, which were renowned for their high abstraction capability and robust encapsulation, has grown considerably in recent times[38]. These techniques offer several advantages: (i) high flexibility, which facilitates the integration of reusable circuit modules in large-scale designs and simplifies circuit topology design. (ii) high coding efficiency, which allows for the straightforward instantiation and interconnection of diverse circuit modules, while its comprehensive error-checking functionality minimizes the necessity for supplementary electronic design automation (EDA) tools for code verification. (iii) Advanced development tools: notable examples include Chisel and SpinalHDL. In our accelerator design, the selection of SpinalHDL is informed by the decision to leverage the Scala library, with the objective of reducing design complexity[39]. This approach allows users to modify accelerator-relevant parameters in order to adapt different resources without requiring in-depth knowledge of internal details.

Quantization Typically, a CNN is trained with 32-bit floating-point data on a GPU. The latest generation of GPUs is capable of handling 16-bit floating-point formats, yet these remain more complex than fixed-point data formats. The reduction in bit width necessitates the coarse quantization of data. The range of data values observed across different layers in a CNN is typically quite broad. Therefore, a uniform quantization with a fixed-point data format for all layers may result in a significant performance degradation. To address this issue, we propose an 8-bit mixed quantization strategy, with the objective of adapting more effectively to the diverse characteristics of the data. Symmetric quantization is employed for data exhibiting minor variations, thereby

facilitating the process. In contrast, asymmetric quantization is employed for data exhibiting significant variations, thereby ensuring the maintenance of precision while minimizing information loss during quantization. The initial step in our quantization process involves training the network using a floating-point (Float32) data format. After the training phase, we perform a layer-by-layer analysis to gather statistical information about the data. Specifically, we calculate the skewness and range values of the parameters in each layer. These statistics help guide the determination of the optimal threshold values for quantization.For each layer, we evaluate multiple potential threshold positions by quantizing the floating-point data to an 8-bit integer (Int8) format. After quantization, the accuracy of the model is tested to assess the impact of each threshold on the performance. Based on this evaluation, the optimal set of thresholds that minimizes accuracy loss is selected and stored for each layer.In some cases, a specific layer may exhibit sensitivity to quantization, meaning it is not well-suited for direct application of this strategy. When this occurs, additional adjustments are made —such as using mixed-precision formats or adjusting the quantization scale—to ensure that there is no significant degradation in accuracy. This process allows us to achieve efficient quantization while maintaining the overall performance of the network. By carefully optimizing the quantization thresholds and making layer-specific adjustments, we are able to reduce the model's bit width from Float32 to Int8 without significantly sacrificing accuracy. This approach ensures that the network can be deployed on resource-constrained hardware platforms while maintaining a high level of performance.The efficacy of this strategy is demonstrated by achieving 74.3% mIoU on the CamVid dataset and 72.7% mIoU on the CityScapes dataset. These outcomes illustrate the capacity of this approach to effectively reduce model size and inference time while maintaining accuracy.

DSP Optimize The most computationally intensive operations in neural networks, such as matrix multiplication and convolution, are typically executed by Digital Signal Processors (DSPs). While INT8 quantization reduces memory storage and bandwidth requirements while maintaining accuracy, the bit width of modern DSPs is significantly larger than the quantized data, leading to inefficiencies in DSP utilization. To address this issue, we implemented specific DSP optimization techniques, including the use of the DSP48E2 module, which is equipped with an 18×27-bit multiplier.For INT8 operands, each DSP48E2 module is capable of computing two dot products in parallel, instead of just one, by processing two sets of INT8 weights and one feature map simultaneously. This parallelism is achieved by concatenating two INT8 weights to align with the DSP bit width, allowing the module to handle twice the amount of data per cycle. In our design, this method efficiently utilizes DSP resources, as matrix multiplication and convolution are highly parallel operations. Generating two dot products with three vectors is a common approach in neural networks, and this optimization significantly accelerates the computation.Through this approach, two dot products are computed simultaneously within a single clock cycle, effectively doubling DSP throughput and reducing the required DSP resources by half. This optimized design not only improves hardware resource utilization but also significantly lowers the power consumption of the accelerator. By processing two computations in parallel within each clock cycle, the overall energy efficiency of the system is greatly enhanced, making it suitable for deployment in resource-constrained environments.

IV. Experiment

A. Experimental Setup and Workflow

The workflow for the designed hardware accelerator is illustrated in Fig.1. In this process, we first utilized the PyTorch framework for model training and quantization. During the actual operation of the hardware accelerator, the FPGA's DRAM was transferred with the quantized data and instructions from the host computer via the PCIe bus. The network inference tasks were executed by the accelerator based on these instructions, and the result data was returned to the host computer via the same PCIe bus upon task completion. It is noteworthy that the core focus of this study was the acceleration of the network inference process, excluding other stages such as image preprocessing and display. Detailed evaluation reports and information were provided in the later sections of the document.

In this section, experiments were conducted based on the aforementioned workflow. The model was trained and quantized using PyTorch 2.2.1 and the CUDA 11.8 toolkit on an NVIDIA RTX3070ti GPU. Finally, the accelerator was deployed on a Virtex UltraScale+ VU9P FPGA, with all hardware implementations developed using SpinalHDL and synthesized and implemented using Vivado 2021.2.

B. Datasets

CamVid This dataset is the first video dataset with object class semantic labels, containing 701 images of urban street scenes, all with a resolution of 960×720. It is divided into a training set (367 images), a test set (233 images), and a validation set (101 images). The dataset includes eleven major feature types: Sky, Building, Pole, Road, Pavement, Tree, SignSymbol, Fence, Car, Pedestrian, and Bicyclist.

CityScapes This dataset focuses on the semantic understanding of urban street scenes, containing images from 50 different cities. It includes 5000 high-quality pixel-level annotated images of driving scenes in urban environments, all with a resolution of 2048x1024. It is divided into a training set (2975 images), a test set (1525 images), and a validation set (500 images). The dataset includes nineteen major feature types: road, sidewalk, building, wall, fence, pole, traffic light, traffic sign, vegetation, terrain, sky, person, rider, car, truck, bus, train, motorcycle, and bicycle.

C. Preprocessing and Hyperparameters

For all images, we resize them to 640x640 during both training and inference. Additionally, for the training set, we use standard data augmentation techniques, including random scaling, random horizontal flipping, normalization, and data shuffling.

The training hyperparameters are as follows: Learning rate scheduling follows a "poly" policy with an initial

learning rate of 0.005.Weight decay is set to $0.5x10^{-4}$.The number of iterations is 20,000.Batch size is 8.Cross-entropy loss is used as the loss function.

D. Quantization Accuracy Evaluation

In this study, we use mIoU as the evaluation metric to measure segmentation performance on the CamVid and CityScapes datasets. The purpose of quantization is to reduce model complexity and hardware resource usage. In the experiments, we tested three different quantization methods, as shown in Table 2, all using INT8 format: symmetric quantization for weights and inputs, asymmetric quantization, and a hybrid quantization combining both symmetric and asymmetric methods. Hybrid quantization strategy performed best on both datasets, achieving mIoU of 74.3% on the CamVid dataset and 72.7% on the CityScapes dataset.

Table 2 Quantization Method Performance

	Accuracy (%)	
Method	CamVid	CityScapes
Original Accuracy	75.4	73.6
Symmetric Quantization	73.5	72.2
Asymmetric Quantization	73.9	72.6
Mixed Precision Quantization	74.3	72.7

In further experiments, as illustrated in the Fig.10, drop in model accuracy is not significant at higher quantization bit widths, mainly because the model size did not significantly reduce. When the quantization bit width drops below 8 bits, the accuracy loss becomes very noticeable. Based on this observation, we selected 8-bit quantization as our strategy. mIoU on the CamVid dataset decreased by 1.6 percentage points compared to 32-bit floating-point computation, and by 1.9 percentage points on the CityScapes dataset. Despite this reduction in accuracy, our model size was reduced by a factor of four. Loss in quantization accuracy is primarily due to clipping and rounding errors during the quantization process, which often conflict with each other. Semantic segmentation tasks are more sensitive to low bit-width quantization compared to image classification tasks, resulting in more pronounced accuracy loss in semantic segmentation.

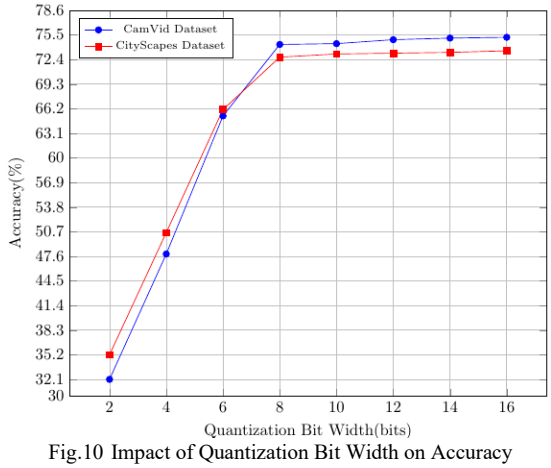

Fig.10 Impact of Quantization Bit Width on Accuracy

E. Comparison of Previous Work

We compare our work with previous studies on two datasets, with detailed results presented in Tables 3 and 4. Yu primarily contributed an 8-bit quantization strategy, which minimized accuracy loss to only 2.04%. In contrast, our post-training quantization method resulted in a mere 1.6% accuracy drop, precision of the OpenCL accelerator was also limited by the network's structure[41]. Miyama used a basic U-Net architecture with quantized weights and activations[27]. Due to the network's simplicity and reduced input image size, low-bit quantization had minimal impact on accuracy, leading to impressive results. However, its practicality is limited due to its narrow applicability. Shimoda focused on optimizing networks with a large number of parameters using filter pruning methods[42]. While effective, the rise of lightweight deep learning models has reduced the relative benefits of this approach. Lightweight models maintain high accuracy with fewer parameters, significantly improving efficiency and applicability.

work	OpenCL accelerator[41]	CamVid 3-bit Quantized CNN[27]	SDCN accelerator[42]	our
Device	Arria-10 FPGA	Alveo U200	zcu102	VU9P
Precision	8-bit quant	3-bit quant	-	8-bit fix-quan
Net	SegNet-basic	Unet	Alex Net-based SFCN	BiSeNetV1
Input size	360x480	256x256	300x225	640x640
Frame rate (FPS)	57	123	165	25
Frequency (MHz)	-	300	100	200
DSPs	-	882	-	1188
mIoU(%)	57.91	67.8	42.62	74.3

In contrast, Jia used the E-Net network and the Xilinx Vitis-AI compiler to convert floating-point models into fixed-point models executed by the DPU. The limitation of their method is its applicability to specific network structures, lacking generalization[43]. Mori adopted the lightweight DeepLabV3+ network, focusing on pruning model parameters. Although it showed significant performance improvement over CPU, it still fell short of real-time requirements[26]. Le utilized a U-Net-based structure with 4-bit quantization, excelling in resource utilization but achieving the lowest accuracy among all compared works, highlighting a trade-off between accuracy and resource usage[44].

work	Real-time FPGA accelerator[43]	CityScapes channel pruning CNN[26]	SDCN accelerator[44]	our
Device	ZYNQ 7035 FPGA	Arria-10 FPGA	Alveo U250 FPGA	VU9P
Precision	8-bit quant	-	4-bit quant	8-bit fix-quant
Net	E-Net	DeepLabV3+	U-Net	BiSeNetV1
Input size	1024x512	960x960	256x256	640x640
Frame rate (FPS)	32.9	1.4	22.6	24
Frequency (MHz)	-	200	152	200
DSPs	689	-	1043	1188
mIoU(%)	63.9	65.29	62.9	72.7

Our method employs the BiSeNetV1 network with an input size of 640x640. On the CamVid dataset, it achieved 25 FPS at a 200 MHz operating frequency with a mIoU of 74.3%. On the CityScapes dataset, it reached 24 FPS and a mIoU of 72.7%. This performance is crucial for practical applications, demonstrating an excellent balance between accuracy and real-time performance. Our work proves to be a

general-purpose solution suitable for a wide range of datasets, meeting high-performance requirements. This dual optimization strategy not only enhances the adaptability of the network but also ensures the efficiency of hardware accelerator deployment.

Part of semantic segmentation results are shown in Fig.11, the two results on the left are from the CamVid dataset, and the two results on the right are from the CityScapes dataset.

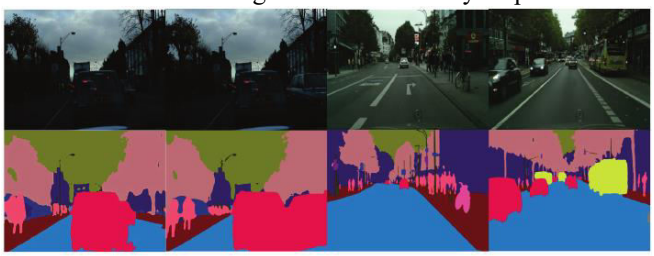

Fig.11 semantic segmentation result samples

V. Conclusion

In this paper, we proposed a reconfigurable semantic segmentation accelerator that addresses the issue of speed and accuracy imbalance in semantic segmentation. The design was rendered more user-friendly by virtue of the fact that SpinalHDL is highly parameterized, thereby facilitating adaptation to a variety of neural networks through a modular design. To address the complexity of scheduling between advanced algorithms and hardware, a lightweight controller was designed for the purpose of controlling the execution of data flow instructions. This had the potential to significantly reduce the time required for subsequent network iterations during the development process.

To reduce inference time, we adopted a mixed quantization strategy, which had proven effective. Additionally, we optimized convolution computations to fully exploit FPGA parallelism. The accelerator was implemented on the Virtex UltraScale+ VU9P FPGA and tested on two common datasets, showing significantly better performance than previous works.

References

[1] Jogin, M., Madhulika, M., Divya, G., Meghana, R., Apoorva, S., et al.: Feature extraction using convolution neural networks (cnn) and deep learning. In: 2018 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), pp. 2319–2323 (2018). IEEE.

[2] Wu, B., Iandola, F., Jin, P.H., Kurtzke, K.: Squeezenet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 129–137 (2017).

[3] Ghielmetti, N., Loncar, V., Pierini, M., Rood, M., Summers, S., Aarestad, T., Petersson, C., Linander, H., Ngdinuba, J., Lin, K., *et al.*: Real-time semantic segmentation on fpgas for autonomous vehicles with hls4ml. *Machine Learning: Science and Technology* 3(4), 045011 (2022).

[4] Hao, C., Chen, Y., Liu, X., Sarwari, A., Sew, D., Dhar, A., Wu, B., Fu, D., Xiong, J., Hwu, W.-m., *et al.*: Nais: Neural architecture and implementation search and its applications in autonomous driving. In: 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 1–8 (2019). IEEE.

[5] Valadarzoj, Z., Daryanavard, H., Harifi, A.: High-speed yolov4-tiny hardware accelerator for self-driving automotive. *The Journal of Supercomputing* 80(5), 6699–6724 (2024).

[6] Guo, K., Sui, L., Qiu, J., Yu, J., Wang, J., Yao, S., Han, S., Wang, Y., Yang, H.: Angel-eye: A complete design flow for mapping cnn onto embedded fpga. *IEEE transactions on computer-aided design of integrated circuits and systems* 37(1), 35–47 (2017).

[7] Du, L., Du, Y., Li, Y., Su, J., Kuan, Y.-C., Liu, C.-C., Chang, M.-C.F.: A reconfigurable streaming deep convolutional neural network accelerator for internet of things. *IEEE Transactions on Circuits and Systems I: Regular Papers* 65(1), 198–208 (2017).

[8] Yu, C., Wang, J., Peng, C., Gao, C., Yun, G., Sang, N.: Bisenet: Bilateral segmentation network for real-time semantic segmentation. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 325–341 (2018).

[9] Yang, Z., Yan, L., Yuan, J.: Design and implementation of driverless perceptual system based on cpu+ fpga. In: 2020 5th International Conference on Control, Robotics and Cybernetics (CRC), pp. 261–265 (2020). IEEE.

[10] Bi, F., Yang, J.: Target detection system design and fpga implementation based on yolov2 algorithm. In: 2019 3rd International Conference on Imaging, Signal Processing and Communication (ICISPC), pp. 10–14 (2019). IEEE

[11] Ahmad, A., Pasha, M.A., Raza, G.J.: Accelerating tiny yolov3 using fpga-based hardware/software co-design. In: 2020 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5 (2020). IEEE

[12] Ma, Y., Cao, Y., Vrudhula, S., Seo, J.-s.: Optimizing the convolution operation to accelerate deep neural networks on fpga. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26(7), 1354–1367 (2018)

[13] Huang, H., Wu, Y., Yu, M., Shi, X., Qiao, F., Luo, L., Wei, Q., Liu, X.: Eddsa: an ecoder-decoder semantic segmentation networks accelerator on opencl-based fpga platform. *Sensors* 20(14), 3969 (2020)

[14] Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3431–3440 (2015)

[15] Chen, L.-C., Papandreou, G., Schroff, F., Adam, H.: Rethinking atrous convolution for semantic image segmentation. arXiv preprint arXiv:1706.05587 (2017)

[16] Zhao, H., Shi, J., Qi, X., Wang, X., Jia, J.: Pyramid scene parsing network. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2881–2890 (2017)

[17] Peng, C., Zhang, X., Yu, G., Luo, G., Sun, J.: Large kernel matters–improve semantic segmentation by global convolutional network. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4353–4361 (2017)

[18] Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: Medical Image Computing and Computer-assisted intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5–9, 2015, Proceedings, Part III 18, pp. 234–241 (2015). Springer

[19] Badrinarayanan, V., Kendall, A., Cipolla, R.: Segnet: A deep convolutional encoder-decoder architecture for image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence 39(12), 2481–2495 (2017)

[20] Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., Yuille, A.L.: Semantic image segmentation with deep convolutional nets and fully connected crfs. arXiv preprint arXiv:1412.7062 (2014)

[21] Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., Yuille, A.L.: Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. IEEE Transactions on Pattern Analysis and Machine Intelligence 40(4), 834–848 (2017)

[22] Wu, Z., Shen, C., Hengel, A.v.d.: Real-time semantic image segmentation via spatial sparsity. arXiv preprint arXiv:1712.00213 (2017)

[23] Chollet, F.: Xception: Deep learning with depthwise separable convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1251–1258 (2017)

[24] Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017)

[25] Li, Z., Xu, H., Liu, Z., Luo, L., Wei, Q., Qiao, F.: A 2.17 µw@120fps ultra-low-power dual-mode cmos image sensor with senputing architecture. In: 2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 92–93 (2022). IEEE

[26] Mori, P., Vemparala, M.-R., Fasfous, N., Mitra, S., Sarkar, S., Frickenstein, A., Frickenstein, L., Helms, D., Nagaraja, N.S., Stechele, W., *et al.*: Accelerating and pruning cnns for semantic segmentation on fpga. In: Proceedings of the 59th ACM/IEEE Design Automation Conference, pp. 145–150 (2022)

[27] Miyama, M.: Fpga implementation of 3-bit quantized cnn for semantic segmentation. In: *Journal of Physics: Conference Series*, vol. 1729, p. 012004 (2021). IOP Publishing

[28] Suda, N., Chandra, V., Dasika, G., Mohanty, A., Ma, Y., Vrudhula, S., Seo, J.-s., Cao, Y.: Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks. In: Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp. 16–25 (2016)
[29] Umuroglu, Y., Fraser, N.J., Gambardella, G., Blott, M., Leong, P., Jahre, M., Vissers, K.: Finn: A framework for fast, scalable binarized neural network inference. In: Proceedings of the 2017 ACM/SIGDA International Symposium on Field-programmable Gate Arrays, pp. 65–74 (2017)
[30] Umuroglu, Y., Fraser, N.J., Gambardella, G., Blott, M., Leong, P., Jahre, M., Vissers, K.: Finn: A framework for fast, scalable binarized neural network inference. In: Proceedings of the 2017 ACM/SIGDA International Symposium on Field-programmable Gate Arrays, pp. 65–74 (2017)
[31] Gysel, P., Motamedi, M., Ghiasi, S.: Hardware-oriented approximation of convolutional neural networks. arXiv preprint arXiv:1604.03168 (2016)
[32] Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149 (2015)
[33] Ali, N., Philippe, J.-M., Tain, B., Coussy, P.: Generating efficient fpga-based cnn accelerators from high-level descriptions. *Journal of Signal Processing Systems* 94(10), 945–960 (2022)
[34] Venieris, S.I., Bouganis, C.-S.: Fpgaconvnet: Mapping regular and irregular convolutional neural networks on fpgas. *IEEE Transactions on Neural Networks and Learning Systems* 30(2), 326–342 (2018)
[35] Chen, Y.-H., Emer, J., Sze, V.: Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *ACM SIGARCH Computer Architecture News* 44(3), 367–379 (2016)
[36] Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M.A., Dally, W.J.: Eie: Efficient inference engine on compressed deep neural network. ACM SIGARCH Computer Architecture News 44 (3), 243 - 254 (2016)
[37] Taye, M.M.: Theoretical understanding of convolutional neural network: Concepts, architectures, applications, future directions. Computation 11 (3), 52 (2023)
[38] Ding, C., Gu, J., Du, Y., Han, B., He, H., Hu, Y., Liu, L., Wei, S., Yin, S.: A reconfigurable 2d-mesh noc design with agile development technique of spinalhdl. In: 2023 International Symposium of Electronics Design Automation (ISEDA), pp. 142 - 145 (2023). IEEE
[39] Nigade, A., Pawar, S., Banerjee, A., Das, N., Ghosh, S.: Processor using risc-v isa (2021)
[40] Fu, Y., Wu, E., Sirasao, A.: 8-bit dot-product acceleration. Xilinx Inc.: San Jose, CA, USA, 20 (2017)
[41] Yu, M., Huang, H., Liu, H., He, S., Qiao, F., Luo, L., Xie, F., Liu, X.-J., Yang, H.: Optimizing fpga-based convolutional encoder-decoder architecture for semantic segmentation. In: 2019 IEEE 9th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER), pp. 1436 - 1440 (2019). IEEE
[42] Shimoda, M., Sada, Y., Nakahara, H.: Filter-wise pruning approach to fpga implementation of fully convolutional network for semantic segmentation. In: Applied Reconfigurable Computing: 15th International Symposium, ARC 2019, Darmstadt, Germany, April 9 - 11, 2019, Proceedings 15, pp. 371 - 386 (2019). Springer
[43] Jia, W., Cui, J., Zheng, X., Wu, Q.: Design and implementation of real-time semantic segmentation network based on fpga. In: Proceedings of the 2021 7th International Conference on Computing and Artificial Intelligence, pp. 321 - 325 (2021)
[44] Le Blevec, H., Léonardon, M., Tessier, H., Arzel, M.: Pipelined architecture for a semantic segmentation neural network on fpga. In: 2023 30th IEEE International Conference on Electronics, Circuits and Systems (ICECS), pp. 1 - 4 (2023). IEEE