Towards an Efficient Accelerator for DNN-based Remote Sensing Image Segmentation on FPGAs

Shuanglong Liu*, Xinyu Niu† and Wayne Luk*
*Dept. of Computing, Imperial College London, London, United Kingdom
{*s.liu13, w.luk*}*@imperial.ac.uk*
†Corerain Technologies Ltd., Shenzhen, China, *xinyu.niu@corerain.com*

***Abstract*—Among popular techniques in remote sensing image (RSI) segmentation, Deep Neural Networks (DNNs) have gained increasing interest but often require high computation complexity, which largely limit their applicability in on-board space platforms. Therefore, various dedicated hardware designs on FPGAs have been developed to accelerate DNNs. However, it imposes difficulty on the design of efficient accelerator for DNN-based segmentation algorithms, since they need to perform both convolution and deconvolution which are two fundamentally different types of operations. This paper proposes a uniform architecture to efficiently implement both convolution and deconvolution in one vector multiplication module. This architecture is further optimized through exploiting different levels of parallelism and layer fusion to achieve low latency for RSI segmentation tasks. Moreover, an optimized DNN model is developed for real-time RSI segmentation, which shows preferable accuracy compared to other methods. The proposed hardware accelerator efficiently implements the DNN model on Intel's Arria 10 device, demonstrating 1578 GOPS of throughput and 17.4 ms of latency, i.e., 57 images per second.**

I. Introduction

The high-resolution remote sensing images (RSIs) acquired from space satellites contain rich and large-scale information of the earth's surface and enable both short-term and long-term environmental change detection, which is critical for protecting natural and human environments [1]. In remote sensing, semantic segmentation is often viewed as an important tool for landscape change detection and land use/cover classification, which is widely used in applications such as environmental monitoring [2], disaster assessment [3], surveying and mapping [4]. Methods of image segmentation become more and more difficult to classify the detail information accurately in the field of RSI analysis, due to the increasing spatial resolution of imagery [5].

Recently, Deep Neural Networks (DNNs) have become widely adopted in image segmentation problems and shown great accuracy improvement [6]. However, these DNN-based methods depend on very large computational complexity, in particular when processing RSIs with high spatial resolution. It severely limits their deployment in resource-limited space platforms, since real-time RSI segmentation is needed to reduce download bandwidth and storage requirements in satellite on-board processing [7].

The support of the United Kingdom EPSRC (grant numbers EP/L016796/1, EP/N031768/1, EP/P010040/1 and EP/L00058X/1), Corerain, Maxeler, Intel, Xilinx and SGIIT is gratefully acknowledged.

GPGPUs with highly parallel architectures can achieve high throughput on DNN models by batch processing to offer a high degree of parallelism. The efficiency of GPGPUs largely relies on the batch size, which works well for off-line training but not practical for real-time inference [8]. For streaming applications, images often arrive one at a time and using batch processing can greatly increase latency, which is critical to the system's performance especially for satellite applications [9]. Besides, GPGPUs are not realistic to be mount on a satellite or a drone because of their high power consumption, and therefore their usability is very limited in space platforms [10].

FPGAs have shown very good performance for DNN-based segmentation acceleration [6], [11], and thus are considered promising to accelerate RSI segmentation methods in space platforms. The main challenge and limitations of previous designs of segmentation on FPGA are that: 1) unlike classification or detection networks such as VGG-16 or AlexNet, segmentation networks consist of both convolutional (Conv) and deconvolutional (Deconv) layers which require different mathematical operations, making it difficult to design efficient architectures for them; 2) Deconv must insert zeros into the input image when implemented as Conv, leading to the sparsity of input as well as computational inefficiency due to multiplications with zeros [11].

In this paper, we propose a novel and uniform architecture to efficiently implement both Conv and Deconv layers with arbitrary kernel size in one computation module. We exploit the sparseness of deconvolution by only computing the overlapping weights and non-zero input pixels to improve the computation efficiency. The performance of the proposed accelerator is further optimized through layer fusion, exploiting different levels of parallelism and fully leveraging the DSPs on FPGA device. The main contributions of this work are:

1) A uniform architecture based on vector multiplication to implement both Conv and Deconv with arbitrary kernel size efficiently. With this architecture, we propose an accelerator by caching all intermediate feature maps in on-chip buffers to achieve low latency for real-time RSI segmentation;

2) An optimized DNN model which is developed with reduced computational complexity and improves the trade-off between accuracy and latency, and it is implemented on our accelerator achieving high throughput and low latency;

3) Evaluation of the hardware accelerator running the proposed model on Intel's Arria 10 FPGA device, demonstrating

1578 GOPS of throughput and 17.4 ms of latency, i.e., 57 frames per second (fps).

II. Background and Related Work

A. DNN-based RSI Segmentation

Semantic segmentation is used for partitioning a remote sensing image (RSI) into many regions with homogeneous properties that faithfully correspond to the objects [12]. The characteristics of the pixels in the same region are similar, and the characteristics of pixels in different regions are different. Image segmentation has been playing a particularly important role in image processing and computer vision. However, it is difficult to classify the detail information accurately in RSIs since they contain a large amount of information. In this context, Deep Neural Network (DNN) appears as an appealing alternative to traditional shallow classification approaches to deal with such a massive amount of data [13]. The popular DNN models for semantic segmentation include SegNet, U-Net, DeepLab, FCN, etc. These models are much more computationally intensive than the classification network models such as LeNet, AlexNet and GoogleNet [6].

DNN-based segmentation algorithms often consist of two major computational layers: 1) **Conv** layer, which aims to interpolate the most relevant information from the input image by convolving the input feature maps with the convolution kernel and producing one output feature map; 2) **Deconv** layer, also known as up-sampling or transposed convolution, which transforms the input in the opposite direction of a Conv layer but extrapolates new information from the input feature map. These two layers perform fundamentally different mathematical operations and have different data access patterns. Besides Conv and Deconv layers, **Concat** layers are also performed to concatenate the outputs from the corresponding part of Conv and Deconv layers. Concat layers are not computationally intensive as Conv and Deconv, but impose communication overhead on the implementation.

B. Related Work

Most of existing FPGA-based DNN accelerators target at the performance of convolution by computation reduction techniques such as using fast Fourier transform (FFT) [14] and winograd algorithm [15]. Layer fusion methods are also used to accelerate CNNs in previous work [16], [17] by caching all intermediate feature maps in on-chip memories to reduce the external bandwidth requirements. These accelerators are only focused on Conv layers and may not lead to high performance for Deconv layers as well as networks for segmentation.

Few works have focused on accelerating DeConv layers and generative networks (GANs) [11], [18], [19]. Yazdanbakhsh *et al.* [18] proposed an end-to-end FPGA accelerator for GANs that combined MIMD and SIMD models while separating data retrieval and data processing units at the finest granularity. Yan *et al.* [19] proposed a dual convolution mapping method to make full use of the available computational resources. Liu *et al.* [11] proposed a novel layer fusion method for GANs with a memory-efficient architecture to significantly reduce off-chip data transfers. However, these researchers addressed the accelerations of deconvolution in generative adversarial networks and didn't aim at segmentation problems.

The most relevant work to this paper is presented in [6], which optimized the computation of both Conv and Deconv layers for image segmentation. [6] proposed an efficient method to deal with the computational inefficiency occurred by deconvolution and used the design space exploration methodology to achieve the optimal resource allocation between Conv and Deconv modules. However, their design utilized separate modules for Conv and Deconv, which didn't share the DSPs for multipliers and led to the problem of under-utilization of resources.

Our approach differs from [6] in three aspects: 1) we implement both Conv and Deconv in one vector multiplication module by proposing a uniform architecture, in order to make full utilization of the available DSP resources (Section III-A); 2) we use layer fusion method by caching all feature maps in on-chip buffers to reduce off-chip communication (Section III-D). Therefore, our architecture is compute-bound, such that we can efficiently use all the DSP resources, and ensure that they are occupied in the majority of the time; 3) we exploit channel parallelism instead of data parallelsim in [6] to boost the performance of our accelerator, since channel parallelism is much more flexible and efficient for FPGA implementation compared to data parallelism[1].

III. Accelerator Architecture

This section starts from the uniform architecture proposed to implement Conv and Deconv layers, then presents the overall accelerator design for the segmentation DNN model. Finally we describe the optimization techniques we proposed to improve the performance.

A. The Uniform Architecture

In DNNs, as shown in Code 1, both Conv and Deconv layers take $C \times H_i \times W_i$ feature maps as the inputs, and have the Conv/Deconv kernels ($F \times C \times K \times K$). Each kernel ($K \times K$) is to be convolved or deconvolved with one channel of input ($H_i \times W_i$) by a sliding stride of S to generate an $H \times W$ output map; then the results of C channels are accumulated to produce one filter of output (channel loop in line 2). This process is repeated for F times to produce all filters of the output feature maps ($F \times H \times W$) (filter loop in line 1). The 2-D convolution is described in line 5 of Code 1.

Deconvolution is also called transposed convolution or fractionally strided convolution [20]. However, transposed convolution needs to insert zero paddings into the original input feature maps before implementing convolution, and thus leads to computation inefficiency. In this work, we use the 2-D Deconvolution method proposed in [6] for hardware implementation, which is illustrated in Figure 1. This method exploits the sparseness of deconvolution by multiplying the weight kernel with each input pixel and summing the overlapping area in output maps. This method reduces the total

[1]Details are explained in Section III-B.

Code 1 Convolution and Deconvolution Algorithms

Input: Input feature map $\mathbf{I}$ of shape $C \times H_i \times W_i$;
Weight matrix $\mathbf{W}$ of shape $F \times C \times K \times K$;
Output: Output feature map $\mathbf{O}$ of shape $F \times H \times W$;

1: for $(f = 0; f < F; f++)$ // **filter loop**
2: for $(c = 0; c < C; c++)$ // **channel loop**
3: for $(h = 0; h < H; h++)$ // **row loop**
4: for $(w = 0; w < W; w++)$ // **column loop**
5: // convolution:

$$\mathbf{O}[f][h][w] \mathrel{+}= \sum_{i=1}^{K-1} \sum_{j=1}^{K-1} \mathbf{W}[f][c][i][j] * \mathbf{I}[c][h * S + i][w * S + j]$$

6: // deconvolution:

$$\mathbf{O}[f] \mathrel{+}= deconv(\mathbf{I}[c], \mathbf{W}[f][c])$$ // see Figure 1.

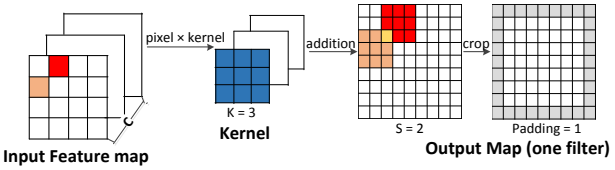


Fig. 1. Visualization of a Deconv layer for $K = 3$, $S = 3$, $Padding = 1$.

multiplications of Deconv from $K^2 \cdot F \cdot C \cdot H \cdot W$ to $r \cdot F \cdot C \cdot H \cdot W$, where $r \in [1, 2)$. More details of this method can be found in [6].

Previous hardware accelerators such as [6], [21] utilized $K \times K$ multipliers to implement the 2-D convolution, i.e., unroll the dot-product loop in line 5. However, this type of architecture cannot be reused for Deconv operation because of the different data pattern for multiplication. Besides, it is also difficult to be reconfigured for Conv kernels with different size (such as 3×3, 5×5 and 7×7).

In this work, we propose a uniform architecture to implement both Conv and Deconv with arbitrary kernel size, by the way that each multiplier is responsible for generating a single pixel of an output image. That is to say, the 2-D convolution or deconvolution for one output pixel is performed in a single multiply and accumulate (MAC) unit in the computation engine, as shown in Figure 2. The uniform architecture shown in Figure 2 contains a vector multiplication module which processes multiple channels of data in parallel, and a quantization module (computing the sum of the input pixels) in support of the 8-bit linear quantization scheme proposed in [22] for integer-only arithmetic inference without accuracy loss. The details about how Conv and Deconv are implemented in this uniform architecture are as follows:

Conv Layer: For each output pixel, the corresponding $K \times K$ input pixels from the input maps and $K \times K$ weights from kernel are multiplied in one multiplier in sequential manner. The results of the multiplier are then accumulated to generate one output of one channel and one filter. Therefore, it takes

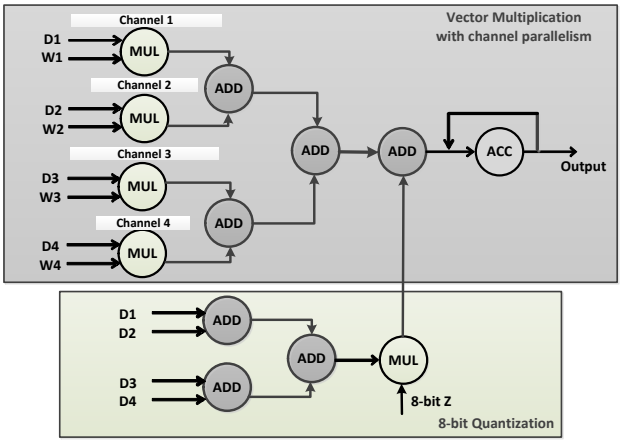


Fig. 2. The uniform architecture to implement Conv and Deconv with 8-bit quantization scheme [22] and multiple channel processing in parallel.

$K \times K$ cycles in total to produce one convolution result. The architecture can implement convolutions with any kernel size and stride values by fetching the right data from input data buffers and feeding them into the vector multiplication module.

Deconv Layer: 2-D Deconv is more complex than Conv. The multiplications and accumulations required for generating one output pixel depend on the position of the output, i.e., how many overlapping rows and columns in the output maps, as we can see from Figure 1. There are three cases in total: 1) for output with non-overlapping, only one input pixel is multiplied by the corresponding weight from kernel; 2) for output with only overlapping row or column, two adjacent input pixels in row or column dimension are multiplied by the corresponding weights in sequence, and the results are accumulated. 3) for output with both overlapping row and column, four adjacent input pixels in row and column dimensions are multiplied by the corresponding weights in sequence, and the results are accumulated. Therefore, it can take 1, 2 or 4 clock cycles to produce one deconvolution result. The deconvolution is implemented in the architecture by fetching the right input data and weights from buffers to be processed by the multipliers.

The proposed uniform architecture can actually support most computation functions in DNN models such as dilated convolution and 1-D convolution, by providing the right sequence of data and weights, and feeding them into the multipliers from data and weight buffers.

B. Parallelism Exploration

To boost the performance of the accelerator, we explore different levels of parallelism in our architecture. There are three levels of parallelism which can be utilized for parallel processing: filter parallelism, channel parallelism and data parallelism. They correspond to unrolling the loop in line 1, 2 and 3 of Code 1. Previous design in [6] used the data parallelism. However, in practical hardware design, we found that data parallelism has a few disadvantages and limitations which lead to computation inefficiency:

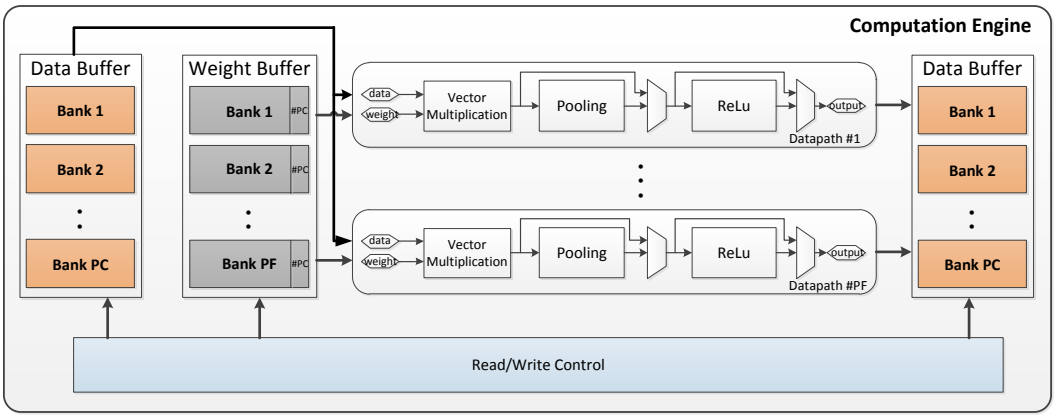


Fig. 3. The over architecture of our accelerator with double buffers for layer fusion, which supports parallelism in channel (PC) and filter (PF) dimensions.

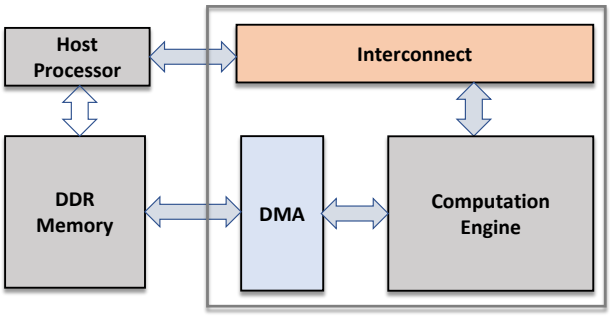


Fig. 4. Our accelerating system consisting of the computation engine, off-chip DDR memories and the host processor.

1) **Workload imbalance** for Deconv implementation. When doing data parallelism, it needs to process multiple output data of one row in parallel. As we have mentioned, Deconv requires different cycles to produce the outputs in one row. That means, every multiplier's workload is imbalanced and some multipliers need to stay idle to wait others finish the processing, so the multiplier utilization will be very low.

2) **Inefficiency** for layers with width which cannot be divided by the degree of parallelism. In hardware design, the degree of parallelism must be a fixed number, e.g. 32. However, in DNNs, the Conv and Deconv layers often have different heights and widths of input maps, and it is impossible to have a degree of parallelism which all the widths of layers in the network can be divided by. Besides, there are layers with small width. For example, for $W = 36$, the computation efficiency is only $\frac{36/32}{\lceil 36/32 \rceil} = 56\%$.

In this work, we utilize channel parallelism (which was not used in [6]) instead of data parallelism, as shown in Figure 2. Multiple channels (PC) of input feature maps are processed in parallel and the results of multiplications are added together using an adder tree before the accumulation unit. The benefit is that every multiplier's workload is balanced, since the outputs of different channels have the same position in output maps. Actually they are producing the same output by addition. Second, the channel (C) of layers (except the first layer) in DNNs are often a power of two, or can be tuned to be a power of two, so they can be divided by the degree of channel parallelism (PC) which is normally a power of two in hardware design. Therefore, it doesn't lead to any loss in the utilization of multipliers and it is very flexible for algorithm development in software and memory system design in hardware. On the contrary, once the size of the first layer's input is determined, the size of all other layers are decided, while the channel values are independent among different layers.

C. Overall Accelerator

Furthermore, our computation engine supports parallelism in filter dimension (PF). The overall accelerator is shown in Figure 3. Each data path contains one vector multiplication module, i.e., the uniform architecture shown in Figure 2, Pooling and ReLu modules. The Pooling and ReLu modules can be bypassed in each data path for flexible layer configuration. To enable parallel filter processing, PF data paths have been instantiated, with each to generate the outputs of F/PF filters. Input data buffers are divided into PC memory banks, and each bank has a memory width of one data pixel with the depth of $H \cdot W \cdot \lceil \frac{C}{PC} \rceil$. Input data are the same for each of all the data paths. Weight buffers are divided into PF memory banks and each bank has a memory width of PC data with the depth of $K \cdot K$. Weights of multiple filters are fed into different rows of the data paths. The PF datapaths share the same quantization module since they have the same input data pixels, and the read/write control logic, in order to save resources.

Our accelerating system is shown in Figure 4. It consists of the computation engine, on-/off-chip interconnect (DMA), off-chip DDR memories and a host processor. All the weights and input map of the first layer, and the final segmentation results are stored in external DDR memory.

D. Layer Fusion

The proposed accelerator utilizes the layer fusion method to reduce the external memory bandwidth requirement by caching all intermediate results in on-chip buffers. Two data buffers shown in Figure 3 are implemented in on-chip RAMs in order to store the input feature data and the output feature data during one layer's execution. These two data buffers have the same data structure and memory size. Before the first convolution layer starts, the images are loaded from the DDR and stored in one buffer. During the convolution layer execution, while the feature data is being streamed into the data paths, the outputs of this layer are simultaneously stored into the second data buffer. When executing the second layer, the second data buffer performs as the input and is being fed into the data paths, the output feature maps will be stored in the other buffer. The final results are loaded back to DDR from data buffers. The double buffer design also reduces the communication overhead when implementing the concat layers, since both input of concat are in on-chip buffers and they are "virtually" concatenated by a proper address generator. It should be noted that double weight buffers are also implemented in our accelerator to overlap the weight load time with the computation time, although this is not shown in Figure 3 for simplicity.

E. Other Optimizations

In order to further boost the performance of our accelerator, we propose a few optimization tricks which are considered in the practical design.

First Layer Execution: The channel of input map of the first layer in DNNs is 3 because of the RGB input image. As we have mentioned, when the channel is smaller than PC or cannot be divided by PC, it leads to computation inefficiency in hardware. This causes performance loss when only 3 of PC multipliers of each data path are working when running the first layer. To address this problem, we divided the input maps into multiple blocks, and these blocks are concatenated along the channel dimension. Correspondingly, the multipliers of the vector multiplication module shown in Figure 2 are grouped by 4 multipliers, and each group takes three channels of data as input and generates one output; the outputs of each data path are PC/4 data in total. In such a way, the computation efficiency of the first layer are improved from 3/PC to 3/4, i.e., 75%.

DSP Configuration: The number of multipliers in FPGA-based accelerators determines the theoretical performance of the system. In FPGA device, multipliers are often implemented using DSPs, making DSP the most limiting resource. The DSP blocks in Intel Arria 10 FPGAs can be fractured into two 18*19 integer multipliers. Our accelerator uses the 8-bit quantization scheme. Even we implement two fixed-point multiplier in one DSP block, it is still a waste of resource when the 18*19 multiplier is used for 8*8 multiplication. We fully leverage the DSPs and logic elements (ALMs) by further implementing one 18*19 multiplier together with some ALM resource as two 8*8 multiplications and one 16-bit addition. To prevent ALMs excessing the limit on device, the DSP blocks are implemented with both two configurations: 1) one DSP for two 8*8 multipliers; 2) one DSP plus ALMs for four 8*8 multipliers and two 16-bit additions. We provide the optimal configuration for DSPs of the system to fully leverage the resource utilization under the constrains of the device.



Fig. 5. Result on the urban surface image from satellites based on our DNN model: input image **(left)** and segmentation result **(right)**.

IV. Evaluation and Experiments

We first propose the DNN model for RSI segmentation and show the results compared to other models, then we evaluate the performance of our accelerator on Intel's Arria 10 FPGA device, with comparison to previous FPGA designs.

A. Proposed DNN Model and Accuracy

In this work, we propose and train DNN models for the segmentation task of remote sensing images obtained from satellites, which is to provide semantic understanding of the urban surface. We propose to optimize the U-Net model originally introduced by Olaf Ronneberger *et al.* in [23], in order to improve the trade-off between accuracy and on-device latency. The proposed model is optimized from three aspects: 1) Width reduction: the number of filter of each layer is reduced by half for thinner models. 2) Resolution multiplier: the input image is adjusted from 572*572 to 256*256 and the internal representation of every layer is subsequently reduced. 3) The unpadded convolutions are replaced by padded convolutions in order to remove the crop operation, and thus the segmentation result has the same resolution as the input image.

The segmentation result for the urban surface images from satellites is shown in Figure 5. The overall accuracy (OA) of the proposed model compared to other models are shown in Table I. Our model provides the overall accuracy of 80.1% which is preferable over most of the other evaluated methods. Under such accuracy, most of the semantics in the image can be understood. Although SegNet and FCN-8s have higher

accuracy than our model, these two models have larger computation complexity which will increase the on-board processing latency for RSI segmentation.

TABLE I
CLASSIFICATION ACCURACY (%) COMPARISON OF THE PROPOSED MODEL AND OTHER METHODS

	SegNet	DeepLab_v3	FCN-32s	FCN-16s	FCN-8s	Proposed (U-Net)
OA (%)	81.7	74.4	76.1	79.1	81.0	80.1

B. Hardware Implementation

We evaluate our accelerator by implementing the proposed model on Intel's Arria 10 SOC FPGAs which contains a A10-SX 660 device (20nm), 1.5 GHz dual-core ARM-based CPU and 2GB DDR4 memory. The whole system is developed using Verilog HDL and implemented with Quartus Prime Pro 18.1 which performs synthesis and implementation. Our accelerator is configured at 64*64 (PC*PF) and running at 200 MHz clock frequency.

C. Resource Utilization

Table II shows the resource utilization of our proposed accelerator at 64*64 (PC*PF) running on the Arria 10 SX 660 device at 200 MHz. Owning to the proposed DSP configurations, we can leverage the DSP blocks fully, and the resource allocation is balanced between ALMs and DSPs.

TABLE II
RESOURCE UTILIZATION OF THE ACCELERATOR ON ARRIA 10 SX 660.

Resources	ALMs	Reg	DSPs	M20K
Used	170,906	377,142	1,665	1,894
Total	251,680	727,160	1,687	2,131
Utilization	68%	52%	99%	89%

D. Performance Comparison

We execute the proposed DNN model in our accelerator, with the input images of size 256*256*3. The total computational complexity of the DNN model is 27.4 GOP (giga operations). Our accelerator achieves the latency of 17.4 ms, which is 57 images per second and 1578 GOPS of throughput.

Table III shows the comparison of our accelerator against prior FPGA accelerators. It should be noted that only [6] implemented the segmentation model containing both Conv and Deconv, and other accelerators were designed for Conv layers only. Nevertheless, our accelerator outperforms all the other accelerators in terms of both resource efficiency (GOPS/DSP) and energy efficiency (GOPS/W), as shown in Table III. These improvements mainly owe to that in our design, the DSP blocks are fully utilized in device and occupied over most of the execution time. Our accelerator gains very large improvement in the performance and resource efficiency by the factor of 14.7× and 7.8× respectively compared to [6] which run the same task as ours. That is due to the proposed uniform architecture which implements both Conv and Deconv, and thus the introduction of Deconv layers doesn't lead to any performance loss in our design.

The high compute and energy efficiency of our accelerator is obtained from taking the advantages of high parallelism [25] and on-chip memory bandwidth [26] of FPGA architecture, and reducing the off-chip memory bandwidth requirement.

V. CONCLUSION

This paper proposes an accelerating system for real-time on-board processing for RSI segmentation in space platforms. We propose a uniform architecture that can efficiently implement both Conv and Deconv layers in DNN models. This architecture is further optimized by exploiting different level of parallelism, layer fusion method and fully leveraging the DSPs. Our accelerator running the proposed model shows higher performance and energy efficiency than others. Future work will target the proposed accelerator at other DNN benchmark models and automate their development process.

TABLE III
COMPARISON WITH PREVIOUS FPGA ACCELERATORS.

	Ma *et al.* [24] in FPGA 2017	**Aydonat *et al.* [17] in FPGA 2017**	**Guo *et al.* [9] in TCAD 2018**	**Liu *et al.* [6] in TRETS 2018**	**Ours**
DNN Model	VGG-16	AlexNet	VGG-16	U-Net	Optimized U-Net
Platform	Intel A10 1150	Intel A10 1150	Xilinx XC7Z020	Xilinx XC7Z045	Intel A10 660
Frequency (MHz)	150	303	214	200	200
Precision	8-16 bit fixed	16-bit float	8-bit fixed	16-bit fixed	8-bit fixed
#DSP	1518	1518	220	900	1688
Logic (ALMs/LUTs)	427K	427K	53K	218K	250K
Power (W)	45	45	3.5	9.6	32*
Latency (ms)	47.97	not reported	364	58.0	17.4
Performance (GOPS)	645.25	1382	84.3	107	1578
Resource Efficiency (GOPS/DSP)	0.425	0.91	0.38	0.12	0.93
Energy Efficiency (GOPS/W)	14.3	30.7	24.1	11.2	49.3

* The power consumption is measured from the board using a power meter.

REFERENCES

[1] R. L. Dodge and R. G. Congalton, "Meeting environmental challenges with remote sensing imagery." AGI, 2013.

[2] E. C. Barrett, *Introduction to environmental remote sensing*. Routledge, 2013.

[3] Y. Fan, Q. Wen, W. Wang, P. Wang, L. Li, and P. Zhang, "Quantifying disaster physical damage using remote sensing dataa technical work flow and case study of the 2014 ludian earthquake in china," *International Journal of Disaster Risk Science*, vol. 8, no. 4, pp. 471–488, 2017.

[4] J. Iliffe, *Datums and map projections for remote sensing, GIS, and surveying*. CRC Press, 2000.

[5] S. Yin, Y. Zhang, and S. Karim, "Large scale remote sensing image segmentation based on fuzzy region competition and gaussian mixture model," *IEEE Access*, vol. 6, pp. 26 069–26 080, 2018.

[6] S. Liu, H. Fan, X. Niu, H.-C. Ng, Y. Chu, and W. Luk, "Optimizing cnn-based segmentation with deeply customized convolutional and deconvolutional architectures on fpga," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 3, pp. 19:1–19:22, Dec. 2018. [Online]. Available: http://doi.acm.org/10.1145/3242900

[7] S. Wang, X. Niu, N. Ma, W. Luk, P. Leong, and Y. Peng, "A scalable dataflow accelerator for real time onboard hyperspectral image classification," in *Applied Reconfigurable Computing*. Cham: Springer International Publishing, 2016, pp. 105–116.

[8] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. K. Reinhardt, A. M. Caulfield, E. S. Chung, and D. Burger, "A configurable cloud-scale dnn processor for real-time ai," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, June 2018, pp. 1–14.

[9] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping cnn onto embedded fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 35–47, Jan 2018.

[10] S. Liu, R. S. Chu, X. Wang, and W. Luk, "Optimizing cnn-based hyperspectral imageclassification on fpgas," in *ARC*, 2019.

[11] S. Liu, C. Zeng, H. Fan, H.-C. Ng, J. Meng, and W. Luk, "Memory-Efficient Architecture for Accelerating Generative Networks on FPGAs," in *IEEE International Conference on Field Programmable Technology (FPT)*, 2018.

[12] L. He, Z. Peng, B. Everding, X. Wang, C. Y. Han, K. L. Weiss, and W. G. Wee, "A comparative study of deformable contour methods on medical image segmentation," *Image and Vision Computing*, vol. 26, no. 2, pp. 141 – 163, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0262885607001230

[13] A. Ben Hamida, A. Benoit, P. Lambert, L. Klein, C. Ben Amar, N. Audebert, and S. Lefvre, "Deep learning for semantic segmentation of remote sensing images with rich spectral content," in *2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, July 2017, pp. 2569–2572.

[14] C. Zhang and V. Prasanna, "Frequency domain acceleration of convolutional neural networks on CPU-FPGA shared memory system," in *FPGA*, 2017, pp. 35–44.

[15] L. Lu *et al.*, "Evaluating fast algorithms for convolutional neural networks on fpgas," in *FCCM*, 2017, pp. 101–108.

[16] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer cnn accelerators," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2016, pp. 1–12.

[17] U. Aydonat *et al.*, "An OpenCL™ Deep Learning Accelerator on Arria 10," in *FPGA*, 2017, pp. 55–64.

[18] A. Yazdanbakhsh *et al.*, "FlexiGAN: An End-to-End Solution for FPGA Acceleration of Generative Adversarial Networks," in *FCCM*, 2018.

[19] J. Yan, S. Yin, F. Tu, L. Liu, and S. Wei, "Gna: Reconfigurable and efficient architecture for generative network acceleration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2519–2529, Nov 2018.

[20] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *arXiv preprint arXiv:1603.07285*, 2016.

[21] R. Zhao, X. Niu, Y. Wu, W. Luk, and Q. Liu, "Optimizing cnn-based object detection algorithms on embedded fpga platforms," in *ARC*. Springer, 2017, pp. 255–267.

[22] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.

[23] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *In Medical Image Computing and Computer-Assisted Intervention*. Springer, 2015, pp. 234–241.

[24] Y. Ma, Y. Cao, S. Vrudhula, and J.-S. Seo, "Optimizing loop operation and dataflow in fpga acceleration of deep convolutional neural networks," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. New York, NY, USA: ACM, 2017, pp. 45–54. [Online]. Available: http://doi.acm.org/10.1145/3020078.3021736

[25] S. Liu, G. Mingas, and C.-S. Bouganis, "An unbiased mcmc fpga-based accelerator in the land of custom precision arithmetic," *IEEE Transactions on Computers*, vol. 66, no. 5, pp. 745–758, 2017.

[26] S. Liu and C.-S. Bouganis, "Communication-aware mcmc method for big data applications on fpgas," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2017, pp. 9–16.