Pipelined Architecture for a Semantic Segmentation Neural Network on FPGA

Hugo Le Blevec, Mathieu Léonardon, Hugo Tessier, Matthieu Arzel

Pipelined Architecture for a Semantic Segmentation Neural Network on FPGA

Hugo Le Blevec*, Mathieu Léonardon*, Hugo Tessier* and Matthieu Arzel*
*IMT Atlantique, Lab-STICC, UMR CNRS 6285, F-29238 Brest, France
*{hugo.leblevec, mathieu.leonardon, hugo.tessier, matthieu.arzel}@imt-atlantique.fr

***Abstract*—Many machine vision tasks like urban scene-understanding rely on machine learning, and more specifically deep neural networks to provide accurate enough results to make technology like autonomous vehicles possible. FPGAs have proven to be an excellent target for deploying highly parallel, low-latency and low-power deep neural network architectures for embedded and cloud applications. Many FPGA implementations use recursive architectures based on Deep Processing Units (DPUs) for fast and resource-efficient solutions which usually come at the cost of a higher latency. On the other hand, pipelined dataflow architectures have the potential to offer scalable, low-latency implementations. In this work, we have explored implementing a semantic segmentation network as a pipelined architecture and evaluated the achievable performances. Our model, a convolutional encoder-decoder based on U-Net, achieves 62.9 % mIoU on the Cityscapes dataset with a 4-bit integer quantization. Once deployed on the Xilinx Alveo U250 FPGA board, the implemented neural network architecture is able to output close to 23 images per second with 44 ms latency per input. The code of this work is open-source and was released publicly.**

***Index Terms*—semantic segmentation, FPGA, deep learning, scene-understanding**

I. Introduction

Semantic segmentation is a computer vision task aiming to identify and separate objects in an image by attributing a label to each pixel. It has become very popular due to recent interest in applications requiring scene understanding, like autonomous driving or computer-assisted surgery. Similarly to other image processing tasks like classification or object detection, it is implemented by deep neural networks that rely on access to high computing power, generally in the form of one or multiple GPUs. This type of power-hungry hardware, however, is not suitable for applications where hardware cost and energy are at stake. Field-Programmable Gate Arrays (FPGAs), on the other hand, represent very good targets to deploy energy-efficient deep neural networks [1] for embedded, edge, and cloud applications. Therefore, many efforts were put to overcome the challenges of implementing such systems on FPGA, which are exceptionally high for semantic segmentation networks, due to more complex topologies and higher feature maps resolution than those of classification networks. A common solution to ease the FPGA implementation uses programmable engines for deep neural networks, like the Deep Processing Unit (DPU) of Vitis AI, which can be used with most neural network topologies, and offer a resource-efficient accelerator with minimal development time. However, due to their recursive design, they rely on many data transfers that limit the achievable performance. An alternative is to implement a fully pipelined and dedicated architecture on FPGA. This paper aims to evaluate such a pipelined architecture and compare it to recursive ones for semantic segmentation networks. We study the implementation of a custom neural network, based on ResNet-18 and U-Net, using FINN[1], an open-source experimental framework for DNN deployment on FPGA [2], [3] and propose a low-latency architecture implemented on Alveo U250 accelerator card. All our code is open-source and freely available[2].

II. Related work

A. Similar approaches

Previous works on segmenting Cityscapes images on FPGA include an ENet model deployed on Xilinx Zynq 7035 using Vitis AI's DPU by Jia et al. [4]. The obtained architecture has the benefit of providing a good throughput for a relatively low amount of resources but has the drawback of having weights stored outside of the on-chip memory, which generates a lot of data-transfer latency, as shown by the average 720 ms per image when considering data transfer, compared to 30.38 ms for the DPU's processing only. Alternatively, Ghielmetti et al. [5] proposed a network based on ENet with a heterogenous quantization and filter pruning. They used HLS4ML to generate a hardware architecture for their network, which is a framework with a similar approach to FINN. It also uses a library of HLS IPs as building blocks to construct a dataflow architecture with on-chip weights storage. The authors obtained a very light architecture deployed on Xilinx ZCU102 and obtained a very low latency of 4.9 ms for a single image, and 30.6 ms for a batch of 10. However, obtaining such efficient hardware came at the expense of the model accuracy, with only 36.8 % of mIoU on the dataset.

B. Pipelined architectures for deep neural networks

As detailed in [2], FINN is an open-source framework that uses customized HLS hardware blocks to build a scalable heterogeneous pipelined architecture. As such, it diverges from the "one-size-fits-all" approach that can be found in

[1]https://xilinx.github.io/finn
[2]https://github.com/hleblevec/finn-semseg

frameworks based on a DPU like Xilinx Vitis AI and instead proposes individual hardware elements that implement the various operations happening in an artificial neural network and are interconnected by AXI interfaces to form a graph that fits the network's structure. It takes the initial graph of the model in ONNX format and operates various transformations on it to map every node of that graph to a corresponding HLS layer. The parallelism level of each element can be tailored to balance the resource utilization and the latency of each layer. This gives FINN a lot of flexibility that allows it to fully explore the design space to find latency-resources trade-offs adapted to any FPGA target and any throughput requirements. An implementation of a ResNet50 with very high throughput and low latency has been made using FINN [6], which shows the tool's potential to implement large residual networks with pipelined architectures and offer better performance than DPU implementations.

As state-of-the-art semantic segmentation networks tend to have more complex structures than classification networks due to residual elements and potentially long shortcuts requiring large memories — a thorny topic on FPGA, we aim to investigate the possibility to use a FINN-generated pipeline architecture for such a network. This study is detailed in the following sections.

III. Implementation

A. Proposed DNN Architecture

State-of-the-art semantic segmentation models based on HRNet [7] have a structure that is too complex to be implemented as a pipelined dataflow architecture, and the memory requirements would arguably be too high to fit on any available device without the use of external memory. As an example, HRNet-18 (the lightest version) has 21 million parameters, so with a 4-bit quantization, the weights would occupy 84Mb of memory, which is already quite consequent for an FPGA. This is not the only memory requirement, as it is also necessary to store intermediate high-resolution feature maps.

Works like ENet [8] or ERFNet [9] have proposed efficient auto-encoders oriented toward hardware implementation that would make good candidates. However, they rely on the use of dilated convolutions which are not yet fully supported by FINN[3]. This left us with more classical convolutional encoder-decoder architectures, out of which U-Net [10] stands as one of the better-performing ones, thanks to the use of intermediate encoder outputs being used as additional side-inputs in the decoder, which was shown to be very significant in improving the accuracy of semantic segmentation networks [11].

Fig. 1 displays a view of the architecture of the network we propose for implementation with FINN. We chose to use the encoder part of a ResNet-18 because it offers a good accuracy-to-number-of-parameters ratio compared to other popular classification networks [12]. We then proceeded with the usual decoder architecture for a U-Net, composed of upsampling layers and convolutions mirroring the encoder.

[3]As of release v0.8.1.

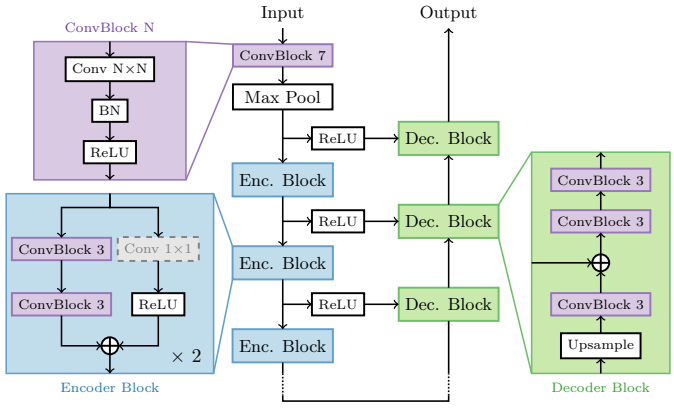

Fig. 1: Representation of the architecture of our semantic segmentation neural network. "BN" stands for Batch Normalization. The element within the Encoder Block is repeated twice each time in the architecture, but we only extract the output of the second for the decoder. As in the original ResNet-18 architecture, each shortcut may contain a downsampling layer in the form of a Convolution with a 1x1 kernel, depending on if the feature map resolution needs to be scaled down. This usually happens only once in each Encoder Block.

We used the Cityscapes dataset to train and evaluate our model [13]. It consists of 5000 images of urban driving scenes with pixel-level fine annotations with 30 visual classes, although only 19 classes are used for benchmarking. The metric used to evaluate models on this dataset is the mean of the IoU (Intersection over Union) across all classes, itself defined as $IoU = \frac{TP}{TP+FN+FP}$ where TP, FN, and FP are respectively the number of true positive, false negative and false positive pixels in the output image with respect to the ground truth. We would normally use transposed convolution to upsample the images but they are not supported by FINN yet, so we had to use nearest-neighbors upsampling instead, which is less accurate than transposed convolutions that have the advantage of having trainable parameters which ensure that the upsampling is relevant to the data. We measured a roughly 3% loss in mIoU after modification. We quantized the model through quantization-aware training with Brevitas [14], using INT4 quantization for weights and activation, except for the first and last layer which are put to INT8, and obtain an mIoU of 62.9% with our model.

We also modified the initial structure by moving ReLU layers from before to after the element-wise additions where the shortcuts reconnect with the main branch and put an additional one after the last convolution. This eases the conversion to HLS layers of FINN without changing the network's accuracy.

Lastly, although FINN does not have an efficient support of DSPs, we moved some Multiply-Accumulate operations of convolutions from LUTs to DSPs to reduce LUT usage and ease the placement step during implementation.

B. FINN Customizations

1) Graph nodes conversion to HLS: In FINN's default build, graph transformations are defined and sequenced in a way that is suitable for networks with relatively "standard" architectures. This sequence is not compatible with residual

networks and with the shortcuts required in our U-Net. Therefore, we used our own custom build within FINN's compiling environment, by changing the order and the nature of the transformations applied to the network, as well as introducing new ones that were not part of the available transformations in the library, making them publicly available in the previously-mentioned repository. These transformations, detailed below, are represented in Fig. 2.

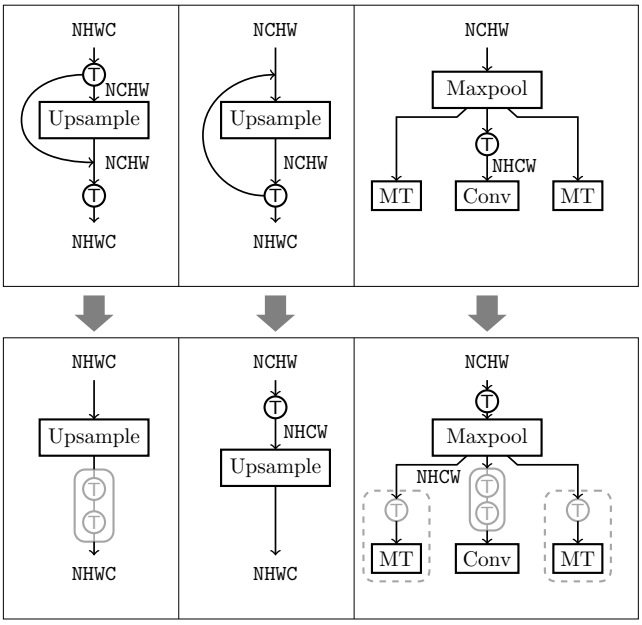

Fig. 2: New proposed transformations in FINN. From left to right: first and second scenarios of data layout conversion for Upsample nodes, and new conversion of MaxPool layers. "MT" stands for MultiThreshold and "T" for Transpose. Two consecutive Transpose can be absorbed together, and a Transpose can be absorbed in a subsequent MultiThreshold.

These transformations are meant to convert the input and output tensors data format from `NCHW` (Number of batches, channel, height, width), which is the usual format of Pytorch to `NHWC` which is the representation used in the HLS library. These transformations were missing or incomplete for Upsample and MaxPool nodes. In the case of Upsample, we have different scenarios depending on the pre-existence of a Transpose node before or after (respectively first and second from the left in Fig. 2). The Transpose node operates the permutation of the dimension and is used to preserve data layout consistency between nodes. The MaxPool data layout conversion (right in Fig. 2) is an update of the already existing one to make it compliant to the case of it being a fork node, which is the case in our model.

2) FIFO insertion: An important step in FINN is the FIFO insertion. Once the network is converted to a graph of HLS IPs, FIFOs must be inserted between certain nodes to ensure that enough data are available to each layer to construct their input. This is essential to ensure that the dataflow architecture can work as a continuous and balanced pipeline, and meet the highest possible throughput. It was necessary to insert very large FIFOs in the shortcuts between the encoder and the decoder part, to make sure the data could be buffered long enough to compensate for the latency of the computations happening in the rest of the architecture. Some of these FIFOs have a depth up to 2^{18} with is 8 times larger than the maximum authorized depth of 2^{15} for FIFO IPs in Vivado, which is used to implement the design. In order to get around this limitation, we developed the *SplitLargeFIFO* transformation within FINN's compiler[4]. The purpose of this operation is to split FIFOs whose depths are over 2^{15} into as many smaller FIFOs as necessary to make sure they are all under this threshold.

IV. Performance measurements

A. Results

We implemented the obtained hardware architecture on Alveo U250 and evaluated its performances using inputs of 256×256 pixels, lower than the original resolution, to meet tools and on-chip memory limitations. The network needs to be fed batches of inputs rather than a single image to reach its best achievable performance due to FINN's pipeline and PCIe interfaces designs [15]. We measured the runtime latency as the time between the moment we send the batch of data to the card, to the time we receive it. As such, it takes into account not just the execution time, but also the data transfer time composed of the PCIe transfer time and the input and output DMA processing time. We carried out measurements with different batch sizes, from 1 to 10^4 by power-of-ten steps. The results are displayed in Fig. 3.

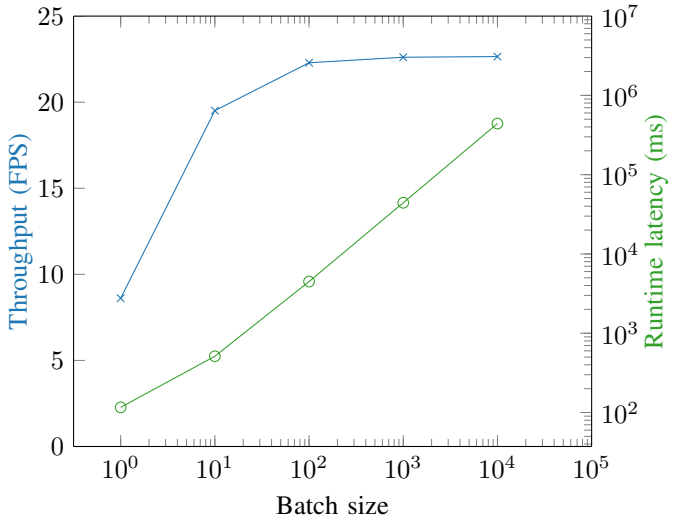

Fig. 3: Semantic segmentation throughput (including the cost of data transfers over PCIe) in frames-per-second (blue, cross) and runtime latency in ms (green, circles) of the network on Alveo U250.

As expected, the throughput increases with the batch size and reaches its maximum value of around 22.6 FPS with a batch of at least 100 images. The latency evolves linearly with the batch size with an overhead corresponding to the interface's latency that is visible for small batch sizes (1 or

[4]Released as a new official feature in FINN v0.9.

Model	HW Arch.	mIoU	Target	Avg. Latency[1]	FPS [2]	FPS* [3]	Clock Rate	LUT	BRAM	FF	DSP
ResNet18-Unet [Ours]	Pipelined	62.9 %	Alveo U250	44 ms	22.65	32.04	152 MHz	490,265	4,260	569,527	1,043
EnetHQ [5]	Pipelined	36.8 %	ZCU102	3 ms	-	-	140 MHz	76.718	224.5	87,059	450
ENet [4]	DPU	63.1 %	Zynq 7035	720 ms	1.39	32.9	-	62,599	257	192,212	689

[1] Average per input. Computed with an input batch of 10 for [5] and over 58 single images for [4]. For the latter, we considered the latency measurement when taking the data transfer into account, which we believe to be a fair comparison to our measurements.
[2] Average system FPS when considering data transfers.
[3] Average on-chip FPS when considering computing time only.

TABLE I: Performance and resources usage comparison between our design and two similar works.

10) but becomes negligible for higher batch sizes. Doing a linear regression gives an intercept of 71 ms and a slope of 44 ms per input.

B. Comparison to previous works

We compared ourselves to the two other similar works presented in Section II-A and show the results in Table I. Although our latency is much higher than EnetHQ [5], our model offers significantly higher accuracy. The implementation of ENet[4] with Vitis AI has a faster segmentation throughput when considering on-chip computations only, but displays a much larger latency due to costly data transfers, which shows the ability of a pipelined architecture to achieve substantially lower latencies than DPUs. We must acknowledge that both these accelerators require much lower resources than ours, and therefore have been able to be deployed on platforms with Zynq devices, which are more suited for embedded applications than Alveo U250, which is essentially a datacenter accelerator board. As the limitations of FINN progressively get lifted with the constant development of new features, the methodology presented in this paper will allow the implementation of a lighter neural network, more suited for embedded deployment, with a pipelined dataflow architecture, and thus obtaining competitive runtime performance while maintaining a good model accuracy on a smaller target.

V. Conclusion

We proposed a pipelined implementation of a 4-bit quantized semantic segmentation network on Alveo U250 FPGA using FINN that achieves 62.9 % mIoU on Cityscapes and is, to our knowledge, the first implementation on FPGA of a UNet-based network segmenting Cityscapes. Even with a complex neural network, our implementation is on par with the state of the art, paving the way to very efficient implementations for real-time semantic segmentation on embedded devices. Our line of work goes now toward embedded deployment with a lighter architecture exploiting the tools and methodology detailed in this paper.

Acknowledgment

We would like to thank the members of the research team of AMD Dublin, and especially Thomas Preußer and Jakoba Petri-Koenig for their precious help throughout the carrying out of this work.

References

[1] X. Xu, X. Zhang, B. Yu, X. S. Hu, C. Rowen, J. Hu, and Y. Shi, "DAC-SDC Low Power Object Detection Challenge for UAV Applications," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 2, pp. 392–403, 2021.

[2] M. Blott, T. B. Preußer, N. J. Fraser, G. Gambardella, K. O'brien, Y. Umuroglu, M. Leeser, and K. Vissers, "FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 11, no. 3, pp. 1–23, 2018.

[3] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. ACM, 2017, pp. 65–74.

[4] W. Jia, J. Cui, X. Zheng, and Q. Wu, "Design and Implementation of Real-Time Semantic Segmentation Network Based on FPGA," in *2021 7th International Conference on Computing and Artificial Intelligence*, ser. ICCAI 2021, 2021, p. 321–325.

[5] N. Ghielmetti, V. Loncar, M. Pierini, M. Roed, S. Summers, T. Aarrestad, C. Petersson, H. Linander, J. Ngadiuba, K. Lin, and P. Harris, "Real-time semantic segmentation on fpgas for autonomous vehicles with hls4ml," p. 045011, nov 2022. [Online]. Available: https://dx.doi.org/10.1088/2632-2153/ac9cb5

[6] L. Petrica, T. Alonso, M. Kroes, N. Fraser, S. Cotofana, and M. Blott, "Memory-Efficient Dataflow Inference for Deep CNNs on FPGA," in *2020 International Conference on Field-Programmable Technology (ICFPT)*, 2020, pp. 48–55.

[7] K. Sun, B. Xiao, D. Liu, and J. Wang, "Deep High-Resolution Representation Learning for Human Pose Estimation," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 5686–5696.

[8] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation," *CoRR*, vol. abs/1606.02147, 2016. [Online]. Available: http://arxiv.org/abs/1606.02147

[9] E. Romera, J. M. Álvarez, L. M. Bergasa, and R. Arroyo, "ERFNet: Efficient Residual Factorized ConvNet for Real-Time Semantic Segmentation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 1, pp. 263–272, 2018.

[10] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," *CoRR*, vol. abs/1505.04597, 2015. [Online]. Available: http://arxiv.org/abs/1505.04597

[11] M. Drozdzal, E. Vorontsov, G. Chartrand, S. Kadoury, and C. Pal, "The Importance of Skip Connections in Biomedical Image Segmentation," *CoRR*, vol. abs/1608.04117, 2016. [Online]. Available: http://arxiv.org/abs/1608.04117

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[13] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The Cityscapes Dataset for Semantic Urban Scene Understanding," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[14] A. Pappalardo, "Xilinx/brevitas," 2022. [Online]. Available: https://doi.org/10.5281/zenodo.3333552

[15] M. Blott, N. J. Fraser, G. Gambardella, L. Halder, J. Kath, Z. Neveu, Y. Umuroglu, A. Vasilciuc, M. Leeser, and L. Doyle, "Evaluation of Optimized CNNs on Heterogeneous Accelerators Using a Novel Benchmarking Approach," *IEEE Transactions on Computers*, vol. 70, no. 10, pp. 1654–1669, 2021.