Real-Time Semantic Segmentation of Aerial Images Using an Embedded U-Net: A Comparison of CPU, GPU, and FPGA Workflows

Julien Posso*, Hugo Kieffer^{†‡}, Nicolas Menga^{†§}, Omar Hlimi[†], Sébastien Tarris^{†‡},

Hubert Guerard[¶], Guy Bois^{*¶}, Matthieu Couderc^{†§}, Eric Jenn[†]

* École Polytechnique de Montréal

[†] IRT Saint Exupéry

[‡] Viveris Technologies

[§] Airbus Defence and Space

¶ Space Codesign Systems

Abstract—This study introduces a lightweight U-Net model optimized for real-time semantic segmentation of aerial images, targeting the efficient utilization of Commercial Off-The-Shelf (COTS) embedded computing platforms. We maintain the accuracy of the U-Net on a real-world dataset while significantly reducing the model's parameters and Multiply-Accumulate (MAC) operations by a factor of 16. Our comprehensive analysis covers three hardware platforms (CPU, GPU, and FPGA) and five different toolchains (TVM, FINN, Vitis AI, TensorFlow GPU, and cuDNN), assessing each on metrics such as latency, power consumption, memory footprint, energy efficiency, and FPGA resource usage. The results highlight the trade-offs between these platforms and toolchains, with a particular focus on the practical deployment challenges in real-world applications. Our findings demonstrate that while the FPGA with Vitis AI emerges as the superior choice due to its performance, energy efficiency, and maturity, it requires specialized hardware knowledge, emphasizing the need for a balanced approach in selecting embedded computing solutions for semantic segmentation tasks.

Index Terms—Deep Learning, Neural Networks, Computer Vision, Semantic Segmentation, Inference, Embedded Systems, Aerospace, CPU, GPU, FPGA, MPSoC

I. INTRODUCTION

The advent of deep neural networks, especially Convolutional Neural Networks (CNNs), has revolutionized computer vision [13], introducing advanced capabilities for embedded systems in areas such as autonomous navigation [29] and earth observation [7], [16], [20]. Efficient hardware acceleration is vital for leveraging this technology, involving CPUs, GPUs, ASICs, FPGAs [27], and neural network compilers that bridge the gap between high-level Python libraries and hardware accelerators [5]. These topics have recently gained significant attention, as discussed in Section II. However, prior research has predominantly focused on image classification networks, specific hardware platforms, and compilers.

In this article, we present a pioneering, comprehensive, transversal study on the optimized implementation of image segmentation tasks for UAVs (Unmanned Aerial Vehicles) and satellites: specifically, the semantic segmentation of aerial images. We have enhanced a U-Net model for improved

embeddability, reducing its parameters and MAC (Multiply-Accumulate) operations by a factor of 16 while maintaining accuracy. We evaluate and compare five implementation schemes (workflows) across three COTS (Commercial Off-The-Shelf) embedded computing platforms (GPU, CPU, FPGA), assessing them using metrics such as IoU (Intersection over Union), accuracy, power, throughput, energy efficiency, and memory footprint. We also consider engineering metrics like workflow maturity, usability, documentation, and community support. This study addresses key practical challenges and provides valuable insights for those looking to integrate deep neural networks into real-world applications.

The structure of this paper is organized as follows: Section II reviews the literature pertinent to our research, providing foundational context. Section III details our computer vision task, specifically focusing on the semantic segmentation of aerial images using a lightweight U-Net to enhance its suitability for embedded systems. Section IV discusses the embedded computing platforms and examines the five workflows employed for implementing the neural network on these platforms. Section V synthesizes the main results, compares the workflows, and discusses the limitations of our study. Finally, Section VI summarizes the study, highlighting the effectiveness of the workflows and the suitability of the hardware selections for our specific application domain.

II. RELATED WORKS

The quest for hardware accelerators is crucial for enabling real-time neural network inference. Central to this acceleration are technologies such as CPUs, GPUs, ASICs, and FPGAs [27]. The role of compilers in bridging the gap between hardware capabilities and neural network performance is well-documented [5]. Additionally, there is a noticeable shift in the embedded sector towards the adoption of Commercial Off-The-Shelf (COTS) computers [24].

Zhao et al. [35] and Li et al. [14] meticulously review prevalent neural network compilers, including TVM, focusing

primarily on their optimization mechanisms and their impact on the speedup of state-of-the-art image classification networks. Xing et al. [33] provide an in-depth analysis of throughput, energy efficiency, and user-friendliness of six compilers, including TVM, aligning closely with our research. However, their analysis is confined to image classification networks such as ResNet50 and SqueezeNet, and they overlook potential quality degradation in neural network output due to the compilation and optimization processes.

Mittal et al. [18] provide a detailed survey of Nvidia Jetson GPUs within the context of embedded systems, including their application in semantic segmentation networks. Abdelouahab et al. [1] and Guo et al. [9] review designs for neural network accelerators, with a particular emphasis on enhancing FPGA inference within image classification networks. Reuther et al. [27] offer a comprehensive yet succinct survey of machine learning accelerators, focusing on performance and energy efficiency. Peccerillo et al. [26] examine approximately 100 accelerators, exploring their diverse workflows.

Comparative studies on FPGA and GPU inference performance and energy efficiency for standard image classification networks are detailed by Nurvitadhi et al. [22]. Feng [8] compares FPGA and GPU inference, focusing solely on semantic segmentation networks on GPUs, notably excluding FPGAs. Li et al. [15] highlight a performance comparison between FPGA and GPU inferences of binarized neural networks, revealing a trade-off between throughput and energy efficiency.

In the embedded domain, Dimitrovski et al. [7] review neural network architectures for aerial imagery, primarily focusing on image classification accuracy while neglecting real-time inference capabilities. Wang et al. [31] and Wu et al. [32] propose new neural network architectures for real-time semantic segmentation of aerial images, yet their deployment on embedded hardware remains unexplored. Moreover, existing research often limits its focus to single COTS platforms and toolchains for real-time inference [18], [30].

The literature exhibits significant limitations, predominantly focusing on image classification networks, which are less relevant for earth observation via UAVs and satellites. Furthermore, the research largely relies on benchmark datasets (e.g., ImageNet) and often restricts its experimental scope to single COTS platforms and toolchains. Studies encompassing multiple hardware targets or compilers are typically classified as surveys rather than experimental research.

In contrast, our research stands out due to its comprehensive approach in several key areas:

- A focus on semantic segmentation, an essential task for analyzing imagery from UAVs and satellites, diverging from the common focus on image classification.
- The adoption of a U-Net architecture for image segmentation, which includes both down-sampling (encoder) and up-sampling (decoder) paths, contrasting with the solely down-sampling nature of image classification networks. This approach exposes unique challenges in certain workflows that previous studies have not addressed.

- The utilization of the *Inria Aerial Image Labeling Dataset* for real-world applications, moving away from the conventional use of benchmark datasets like ImageNet.
- A comprehensive evaluation involving multiple workflows and hardware targets, providing a holistic view of their performance and limitations.

III. EMBEDDABLE U-NET-BASED SEMANTIC SEGMENTATION OF AERIAL IMAGES

A. Semantic Segmentation of Aerial Images

Our research is situated within the context of earth observation, focusing primarily on two application domains: satellites and UAVs. These platforms are pivotal in acquiring high-resolution terrestrial imagery, offering spatial resolutions ranging from 0.2 to 10 meters, which are critical for numerous remote sensing applications [7], [16], [20]. The primary limitation lies in the downlink capacity, as satellites and UAVs lack the capability to transmit all captured images to ground stations. Consequently, on-board analysis becomes essential to ensure that only relevant data is transmitted to Earth, optimizing both bandwidth and data relevance [10].

In this context, semantic segmentation is indispensable as it enables precise on-board analysis of the high-resolution imagery acquired by satellites and UAVs. We employ the *Inria* Aerial Image Labeling Dataset provided by Inria, renowned for its utility in benchmarking the generalization capabilities of semantic segmentation methodologies [17]. This dataset includes 180 colored satellite photographs, each measuring 5000x5000 pixels (25 Megapixels). The primary task of the dataset involves semantic segmentation, which entails classifying each pixel of an input image into a specific category; in our case, this means distinguishing every pixel as either 'building' or 'not building'. This classification results in a segmentation map. Figure 3 illustrates this process. To optimize for training and model embeddability, we dissect these images into smaller segments of 256x256 pixels, maintaining slight overlaps. These segments are subsequently merged to reconstruct the original 5000x5000 segmentation map postinference.

B. U-Net Architecture

We selected a U-Net architecture for our workflow comparison. The U-Net [28], initially proposed for biomedical image segmentation, has since become a widespread neural network architecture. It features a low number of parameters, a small memory footprint, and fewer MAC operations compared to other semantic segmentation networks, while still maintaining high accuracy. Additionally, it is designed to be trained with a limited amount of data, a common scenario in the embedded domain. These characteristics make the U-Net an ideal candidate for an embedded neural network.

However, we modified the U-Net to enhance its embeddability. We trained multiple versions of the U-Net, varying the number of layers and channels per layer. Figure 1 demonstrates the necessity of this process in an embedded context. In the down-sampling path of the U-Net, each block contains two convolutional layers and one max pooling layer. Similarly, in the up-sampling path, each block includes one transposed convolution and two convolutional layers. We adjusted the number of channels on each layer from 1/32 to 1/2 of the original U-Net and varied the number of blocks (*i.e.* the number of layers) from one to four, while maintaining symmetry between the down-sampling and up-sampling paths of the U-Net.

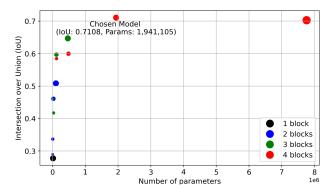


Fig. 1: IoU on the validation set vs. the number of parameters of the U-Net. Circle size represents the number of channels.

We preserved the core structure of the original U-Net, which consists of four blocks, but reduced the number of channels per layer to one-fourth of the original. This adjustment significantly decreased the number of parameters (from 31 million to 1.9 million) and MAC (Multiply-Accumulate) operations (from 55 billion to 3.4 billion) required to process a single 256x256 image, while still maintaining accuracy on the *Inria* Aerial Image Labeling Dataset. The number of parameters and MAC operations is proportional to the square of the number of channels, underscoring the importance of adapting neural network architectures to new datasets, especially in embedded contexts. Figure 2 provides a detailed view of the U-Net architecture, showing the distribution of MAC operations and the number of parameters across the down-sampling (encoder), middle, and up-sampling (decoder) paths. Notably, the two middle layers of the U-Net contain almost half of the parameters, while the majority of MAC operations occur in the up-sampling path. This path is crucial for reconstructing the feature maps back to the original image size, explaining the higher number of MAC operations required for accurately generating the output segmentation map. The inclusion of transposed convolutions in the up-sampling path, not present in state-of-the-art image classification neural networks, introduces unique challenges in some workflows.

C. Training Details

We trained our U-Net on an Nvidia RTX 3070 GPU, using Keras and TensorFlow 2.6, on the *Inria Aerial Image Labeling Dataset*, as detailed in Section III-A. Training began with random initial Float32 weights and utilized the Adam optimization algorithm [11] with TensorFlow's default parameters and a learning rate of 1.0×10^{-4} over 108 epochs. Training

was halted after 15 epochs without improvement in the Intersection over Union (IoU) computed on the validation set. We employed the Binary Cross Entropy (BCE) loss function, which is effective for binary segmentation tasks. To enhance the model's robustness and reduce sensitivity to overfitting, we normalized the input images to a range between 0 and 1 and applied data augmentation techniques using OpenCV 2.5. These techniques included random rotations (multiples of 90 degrees) and horizontal and vertical flipping.

D. U-Net Evaluation

Table I presents the evaluation of our lightweight U-Net, compared with the same data, task, and evaluation metrics used by the Inria team [17]: the IoU of the building class and pixel accuracy. The Inria team employed a FCN (Fully Convolutional Network) followed by a MLP (Multi-Layer Perceptron). Additionally, they discuss the general training process but lack in-depth technical specifics about the architecture configurations, such as the number of parameters and layers. Nevertheless, the evaluations demonstrate that our lightweight U-Net outperforms the Inria team's neural network. The lightweight U-Net serves as a baseline for evaluating the five workflows explored in this paper.

TABLE I: Evaluation metrics of our lightweight U-Net on the validation set

Model	IoU	Accuracy
Lightweight U-Net (ours)	0.7108	0.9546
FCN + MLP (Inria) [17]	0.6467	0.9442

Figure 3 shows an example of our U-Net's prediction quality compared to the ground truth on a 256x256 image. The buildings are generally well-predicted by the neural network, even if the contours of the predicted buildings are somewhat blurred, a similar effect was noticed in the original Inria publication [17].

IV. PLATFORMS AND WORKFLOWS

A. Platforms for Real-Time Inference

We selected two COTS platforms, specifically designed for embedded applications, to deploy our U-Net model. The Xilinx Zynq UltraScale+ MPSoC, equipped with four ARM Cortex-A53 processor cores and programmable logic (commonly referred to as an FPGA), has proven effective in both UAV [12] and space domains [24]. For our implementation, we utilized three Xilinx Zynq UltraScale+ boards—Ultra96, ZCU102, and ZCU104—each equipped with the same processor but featuring varying FPGA sizes, to host the hardware accelerators. Nvidia Jetson platforms have also emerged as strong contenders for real-time inference of neural networkbased vision algorithms, demonstrating applicability in UAV [30] and space domains [2]. Specifically, we employed the Nvidia Jetson AGX Xavier System on Module, which boasts eight ARM Cortex-A57 processor cores and an integrated GPU, enhancing the acceleration of neural network inference.

Path	Block	Layer type	Input channels	Output channels	Kernel size	Output width	Number of parameters	Number of MAC (million)	Number of parameters	Number of MAC (million)	Number of parameters (%)	Number of MAC (%)
	D1	Convolution	3	16	3	256	448	29.4				
	DI	Convolution	16	16	3	256	2,320	152.0				
	D2	Convolution	16	32	3	128	4,640	76.0				
Down	D2	Convolution	32	32	3	128	9,248	151.5	293,520	862.7	15.1%	25.0%
Down	D3	Convolution	32	64	3	64	18,496	75.8	293,520	802.7	15.1%	25.0%
	D3	Convolution	64	64	3	64	36,928	8 151.3				
	D4	Convolution	64	128	3	32	73,856	75.6				
	D4	Convolution	128	128	3	32	147,584	151.1				
Middle	M1	Convolution	128	256	3	16	295,168	75.6	885,248.0	226.6	45.6%	6.6%
Middle	IVII	Convolution	256	256	3	16	590,080	151.1	005,240.0		45.0%	0.076
		Transpose conv.	256	128	2	32	131,200	134.3	_			68.3%
	U1	Convolution	256	128	3	32	295,040	302.1				
		Convolution	128	128	3	32	147,584	151.1				
	Tr	Transpose conv.	128	64	2	64	32,832	134.5				
	U2	Convolution	128	64	3	64	73,792	302.3		,		
Up		Convolution	64	64	3	64	36,928	151.3	762,320	2,354.7	39.3%	
l ob		Transpose conv.	64	32	2	128	8,224	134.7	762,320	J 2,354.7	39.3%	
	U3	Convolution	64	32	3	128	18,464	302.5				
		Convolution	32	32	3	128	9,248	151.5				
	U4	Transpose conv	32	16	2	256	2,064	135.3				
		Convolution	32	16	3	256	4,624	303.0	1			
		Convolution	16	16	3	256	2,320	152.0				
	Final convo	olution	16	1	1	256	17	1.1	17	1.1	0.0%	0.0%
			Total				1,941,105	3,445.2	1,941,105	3,445.2	100.0%	100.0%

Fig. 2: Detailed architecture of the U-Net model

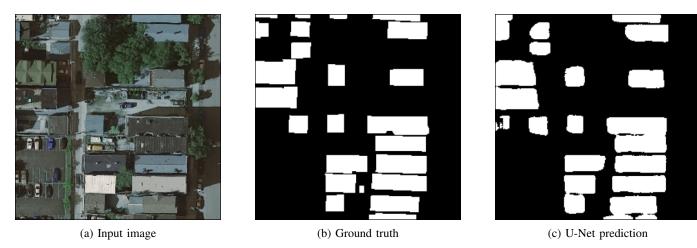


Fig. 3: Qualitative evaluation of our Float32 Keras lightweight U-Net on a 256x256 image of the validation set

B. Workflows Overview

We evaluated various workflows to implement our U-Net on CPU, GPU, and FPGA platforms. On the GPU side, we first assessed the straightforward TensorFlow implementation, comparing it with the more complex but optimized Nvidia cuDNN library to understand the trade-offs between ease of use and performance. For the CPU, we utilized TVM, which is renowned for supporting major Python frameworks and offering the best speedup among neural network compilers [5], further enhanced by its auto-scheduling feature. For the

FPGA, we explored both the open-source FINN framework and Xilinx's commercial DPU within the Vitis-AI toolchain. Although Vitis-AI is considered more mature, FINN offers experimental yet highly optimized options for creating optimized dataflow implementations [3]. The following sections will delve into the details of each workflow.

C. GPU Implementation with TensorFlow

1) Workflow Overview: Figure 4 presents the workflow used to deploy our model on the Nvidia Jetson AGX Xavier

using TensorFlow 2.6. This workflow is straightforward, starting with the training of a Float32 model using Keras, serving as our baseline for evaluating GPU workflows. Notably, the model remains in Float32 format throughout, since quantization is only available in TensorFlow Lite. Our aim was to evaluate the most direct method for deploying a neural network on a Jetson GPU. Furthermore, the Jetson GPU efficiently processes Float32 operations on its CUDA (Compute Unified Device Architecture) cores. The trained model is exported in HDF5 format and then loaded onto the Nvidia Jetson AGX Xavier development kit. Onboard inference is conducted through a Python script, representing the simplest deployment method on the Nvidia Jetson platform, which operates on a Linux-based system with a Python stack, including Tensor-Flow.



Fig. 4: GPU workflow from Keras/TensorFlow training to Nvidia Jetson AGX Xavier inference using TensorFlow

2) Quantitative Evaluation: Table II presents the evaluation metrics measured on the validation set throughout the TensorFlow workflow. The first row shows the results following training with Keras and TensorFlow in a Float32 format. Subsequent rows detail these metrics when the model is deployed on an Nvidia Jetson AGX Xavier board. The consistency observed between the standard computing environment and the embedded deployment is expected because the underlying model remains unchanged between the training and deployment stages.

TABLE II: Evaluation metrics along the TensorFlow workflow

Model	IoU	Accuracy
Float32 Keras	0.7062	0.9594
Jetson implementation	0.7062	0.9594

Table III summarizes the implementation metrics measured on the Jetson AGX Xavier. In this experiment, we varied the batch size to analyze its impact on the implementation metrics. Increasing the batch size to eight proved beneficial for improving throughput and energy efficiency while maintaining a reasonable memory footprint. The memory footprint includes the space needed for the model weights and activation functions, the batch of images, and additional Python libraries such as TensorFlow. Further increases in batch size did not yield significant benefits and resulted in an increased memory footprint, making a batch size of eight an optimal tradeoff. A batch size of one is deemed beneficial only when memory footprint or latency is prioritized over throughput or energy efficiency. During the experiments, we noticed some variability in execution time, particularly for the first inference. The first inference with a batch of eight images took 238

milliseconds, while the subsequent inferences averaged around 107 milliseconds (plus or minus 10 milliseconds). The table also reports the average throughput for the entire validation set. The observed variability was consistent across all batch sizes, highlighting the importance of also considering the Worst Case Execution Time (WCET) in embedded systems where it is a critical factor.

TABLE III: Implementation metrics on the Nvidia Jetson AGX Xavier with TensorFlow

Batch size	Throughput (FPS)	Power (W)	Energy efficiency (mJ/image)	Memory (GB)
1	61.6	13.65	221.6	1.7
8	74.6	14.56	195.2	2.2
16	78.6	14.56	185.2	5.05
32	75.8	14.56	192.1	5.3

3) Qualitative Evaluation: The TensorFlow workflow targeting the Nvidia Jetson GPU is mature, straightforward, and well-documented, supported by an active community with numerous users, examples, and online tutorials. However, optimization of the neural network is limited within this framework. The high memory footprint presents significant concerns for embedded systems, which are often resource-limited compared to typical desktop or server environments. Furthermore, this high memory footprint could impact performance, energy efficiency, cost, and system stability, especially when the hardware is required to manage multiple applications simultaneously.

D. GPU Implementation with CuDNN

- 1) Workflow Overview: Figure 5 illustrates the workflow used to deploy our model on the Nvidia Jetson AGX Xavier utilizing the Nvidia cuDNN 8.4.1 library. Initially, we train a Float32 version of the model using Keras and export the trained parameters. Similar to the previous workflow, the model remains in Float32 format because quantization is only supported in TensorFlow Lite. The Jetson GPU is capable of efficiently processing Float32 operations on its CUDA cores. Subsequently, the neural network must be manually implemented in C++ with calls to the cuDNN library to execute operations on the GPU. The neural network is then cross-compiled for an ARM target using g++ and NVCC (Nvidia CUDA Compiler), resulting in an executable that is deployed on the Nvidia Jetson AGX Xavier, which operates a Linux-based system with the cuDNN library installed.
- 2) Quantitative Evaluation: During the evaluation, we encountered challenges, particularly due to the lack of a cuDNN implementation for the transposed convolution in the upsampling path of the U-Net, as well as for the nearest neighbor upsampling operation. A feasible solution could have been to implement these layers in a custom CUDA program; however, due to limited time and inadequate support on the Nvidia forum, this approach was not viable. We successfully implemented the down-sampling path and the middle convolution of the U-Net using cuDNN. The implementation's accuracy was

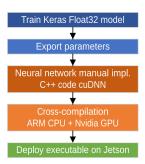


Fig. 5: GPU workflow from Keras/TensorFlow training to Nvidia Jetson AGX Xavier inference with cuDNN

validated by comparing the intermediate tensor outputs from the middle convolution produced by cuDNN with those from TensorFlow, finding them equivalent within an absolute tolerance of 1e-8. Thus, we conclude that the cuDNN workflow is unlikely to alter the evaluation metrics significantly.

Table IV presents the evaluation metrics measured on the validation set for the implemented down-sampling path and middle convolution of the U-Net using cuDNN. We estimated the full U-Net implementation performance by considering that the down-sampling path and middle convolutions comprise 31.6% of the MAC operations, and we scaled the measured latency accordingly to estimate the total latency. Similarly, since these components represent 60.7% of the parameters and intermediate feature maps, we adjusted the memory footprint to estimate the total memory usage. These estimates should be interpreted with caution.

TABLE IV: Measured and estimated implementation metrics on the Nvidia Jetson AGX Xavier with cuDNN

Model	Latency (ms)	Power (W)	Energy efficiency (mJ/image)	Memory (MB)
Partial U-Net (measured)	5.82	5.61	32.6	795
U-Net (esti- mated)	18.4	5.61	103.3	1310

3) Qualitative Evaluation: The cuDNN workflow for targeting Nvidia-embedded GPUs is mature yet intricate. cuDNN is primarily designed for developers of deep neural network (DNN) frameworks such as PyTorch or TensorFlow [4]. Consequently, it is more complex than other libraries and lacks extensive examples. Additionally, the absence of certain neural network layers necessitates a proficiency in CUDA programming, which is considerably more complex than using cuDNN alone. We also encountered discrepancies between the documentation and the actual implementation, which compounded the difficulty. The level of community activity is low; for instance, some queries on the Nvidia forums, particularly concerning transposed convolutions, have remained unanswered for over a year. While cuDNN is the optimal choice for achieving an optimized GPU implementation, especially where the memory footprint is a concern, this advantage requires a significantly greater development effort, particularly for neural networks that include layers not supported by the library.

E. CPU Implementation with TVM

1) Workflow Overview: Figure 6 presents the workflow used to deploy our model on an ARM processor. We began by training a Float32 model with Keras, then utilized TVM 0.8 to export the model to Relay, TVM's intermediate graph representation. At this stage, quantization of the neural network is optional, which we discuss further in section IV-E2. We compiled the model using an optimization level of 3, which in our experiments achieved the best trade-off between optimization and neural network accuracy. Subsequently, we employed TVM's auto-scheduling, conducting 10,000 trials to optimize the scheduling of the inference on the CPU. The model was then ready for deployment on the ARM-A53 target, operating under a Linux-based system with the TVM runtime installed.



Fig. 6: CPU workflow from Keras/TensorFlow training to ARM CPU inference with TVM

2) Quantitative Evaluation: Table V shows the evaluation metrics obtained on a subset of the validation set, consisting of 1500 images, used in the TVM workflow, as the full validation set execution time was prohibitively slow on board. To ensure consistency, we maintained the same sub-validation set from the Keras evaluation through to the onboard evaluation. The table initially reports the metrics following Float32 training with Keras and TensorFlow. Subsequent rows display the metrics obtained when deploying the neural network on an Ultra96 board. The TVM workflow, without quantization, preserved the quality of the neural network's output. In further experiments, we quantized every weight and activation function to eight bits, except for the first convolutional layer. We found that post-training quantization with TVM had a negligible impact on the evaluation metrics, minimally affecting both IoU and accuracy.

TABLE V: Evaluation metrics along the TVM workflow

Model	IoU	Accuracy
Float32 Keras	0.7170	0.9546
Float32 TVM	0.7170	0.9546
Int8 TVM	0.7007	0.9518

Table VI summarizes the performance metrics measured on the Ultra96 and ZCU104 boards. The ZCU104 demonstrated

approximately ten percent faster execution than the Ultra96, attributable to its faster DDR memory. However, latency on both boards was significant, limiting real-time inference of semantic segmentation neural networks on these CPUs. Power consumption averaged 1.1W at thermal equilibrium, which is relatively low and was consistent across both boards and quantization levels, as the ARM cores were fully utilized under all conditions. Quantization increased the execution time threefold, possibly due to a bug in the version of TVM used, suggesting that the auto-scheduling functionality may not be fully compatible with the quantized version of our network. Energy efficiency was slightly better on the ZCU104, but the difference was minimal, except with the quantized version, which showed a significant increase. The memory footprint was reduced further with quantization.

TABLE VI: Implementation metrics on the Xilinx Zynq Ultrascale+ boards with TVM

Board	Quanti- zation	Latency (ms)	Power (W)	Energy efficiency (J/image)	Memory (MB)
Ultra96	No	540.7	1.05	0.568	68.4
Ultra96	Yes	1687	1.05	1.77	39.5
ZCU104	No	489.2	1.11	0.543	78.7

3) Qualitative Evaluation: The TVM workflow for targeting ARM CPUs is well-established, vet it is not without limitations, particularly due to a quantization bug encountered during our evaluations. This issue can be circumvented by utilizing the quantization functionalities of Keras/TensorFlow. The workflow benefits from being user-friendly, supported by extensive documentation and numerous examples. The versatility of the TVM stack allows for deployment on any ARM CPU that operates a Linux-based system, including smartphones and Raspberry Pi devices. Switching the target CPU requires altering only a single line of Python code. The community behind TVM is highly active, annually hosting TVMCon, a conference that fosters collaboration between academia and industry on neural network compilation. TVM's fully automated build and auto-scheduling processes facilitate the deployment and optimization of state-of-the-art convolutional neural networks, rendering the TVM workflow exceptionally adaptable.

F. FPGA Implementation with FINN

1) Workflow Overview: Figure 7 presents the workflow utilized to deploy our model on an FPGA using the FINN library. As FINN is incompatible with Keras or TensorFlow, we re-implemented the U-Net model using PyTorch 1.7.1 and Brevitas 0.6.1. Brevitas is a quantization library designed to facilitate Quantization Aware Training (QAT) with PyTorch and to support deployment through FINN [25]. Initially, we trained a Float32 version of the U-Net using Keras and exported the weights to the PyTorch/Brevitas version of the U-Net. We then proceeded with training a quantized version of the U-Net using QAT in PyTorch/Brevitas, starting from the Float32 weights to significantly reduce QAT duration. Brevitas

supports mixed-precision quantization, enabling layer-wise bitwidth parametrization for both weights and activation functions. After training, the model was exported to the ONNX format, which is compatible with FINN. At this stage, the model is transformed into a graph that contains only FINN HLS-compatible nodes. Subsequently, we defined the folding configuration for each graph node to set the parallelism, aiming to match the target latency without exceeding the FPGA's available resources. If the folding configuration exceeded the FPGA resources, it required returning to the bitwidth parametrization step and reiterating the QAT phase or adjusting the target latency. FINN's built-in functions facilitate the invocation of Vitis HLS to synthesize each node independently, integrate them, and then implement the combined solution as a Vivado 2022.1 project deployed on the FPGA. FINN also offers rapid prototyping capabilities using the Pynq library.

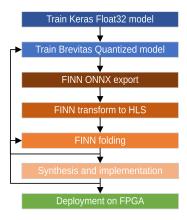


Fig. 7: FPGA workflow from PyTorch/Brevitas training to FPGA inference using FINN

2) Quantitative Evaluation: Table VII displays the evaluation metrics measured on the validation set throughout the FINN workflow. The initial row recalls the metrics after Float32 training with Keras. The final row presents the metrics for the quantized U-Net post-training, employing binary weights and 4-bit activation functions across all layers. Despite aggressive quantization, the accuracy and IoU only showed a slight decrease. Due to a suspected bug in the FINN library, we could not perform onboard inference to directly measure the evaluation metrics, a limitation we will discuss further in section IV-F3.

TABLE VII: Evaluation metrics along the FINN workflow

Model	IoU	Accuracy
Float32 Keras	0.7108	0.9531
Quantized Brevitas	0.6837	0.9488

While onboard inference execution was not possible, we derived certain results from the Vivado project, synthesis, and implementation reports. Table VIII summarizes these findings and estimations. The latency was derived from the synthesis reports, considering the highest latency across all graph nodes

(786,432 cycles) as the accelerator's initiation interval. With a clock frequency of 100 MHz, we estimated the accelerator's latency to be 7.86 milliseconds, corresponding to a throughput of 127 images per second. The on-chip power consumption, estimated at 5.5 Watts, was obtained from the FINN-generated Vivado project. The estimated energy efficiency is noteworthy, given the implementation of a low-bit quantized U-Net, although these results are provisional and should be approached with caution.

TABLE VIII: Estimation of the implementation metrics on the Xilinx ZCU104 with the FINN workflow

Board	Throughput (FPS)	Power (W)	Energy efficiency (J/image)	Memory (MB)
ZCU104	127.2	5.46	0.043	N/A

Table IX provides a summary of FPGA resource utilization based on the post-implementation report generated by Vivado, highlighting LUTs (Lookup Tables) as the primary limiting factor. The LUTs are predominantly utilized for the convolution computations, namely the im2col algorithm and the matrix-vector multiplication unit. Notably, the multi-threshold layers, representing the quantized activation functions, also consume a substantial number of LUTs, proportional to the square of the bit-width of the activation functions. We chose binary weights and 4-bit activations as an optimal balance between accuracy and estimated throughput. This approach also eliminated the need for DSPs, reducing the resource demands significantly. Our experience has shown that the FPGA resource estimations provided by FINN's Python script were found to be unreliable.

TABLE IX: FINN FPGA resource usage on ZCU104 board

FPGA resource	Post- implementation utilization	FINN Python estimation	Available
LUT	205,249 (89%)	155,905	230,400
LUTRAM	43,498 (43%)	Not Available	101,760
Flip-Flop	235,448 (51%)	Not available	460,800
BRAM	96 (31%)	233	312
DSP	0 (0%)	0	1,728

3) Qualitative Evaluation: The Brevitas library for training quantized neural networks targeting FINN implementations is mature and user-friendly, closely mimicking the PyTorch experience, albeit lacking in examples. Conversely, the FINN library is still under development. We encountered and locally fixed several source code bugs during our experiments. While some of these issues have been addressed recently, indicating active development, the community remains relatively small compared to other libraries. The absence of certain HLS backend templates, such as transposed convolution, posed challenges. We circumvented this by substituting with a nearest neighbor upsampling layer followed by a convolution, which did not alter the U-Net's parameter count or MAC operations.

Utilizing the FINN library can be challenging, particularly during the transformation phase, which requires users to

meticulously determine the appropriate transformations and their sequence. Often, modifications to the network architecture and quantization scheme are necessary to remove non-HLS compatible nodes. We had to develop two custom transformations not present in FINN to synthesize the U-Net effectively. A significant issue related to the U-Net's shortcuts prevented us from implementing the neural network on the FPGA. This issue could stem from a problem with our custom transformations, a bug in FINN's handling of concatenation layers, or FINN's algorithm not allocating sufficiently large FIFOs to store the activation functions of the down-sampling path, thereby hampering the up-sampling path's ability to perform its convolutions. Additionally, the documentation, spread across various websites and GitHub pages, is fragmented and challenging to navigate.

The FINN library holds significant potential for energy-constrained applications and is poised to mature into a highly energy-efficient method for executing neural network inference on FPGAs. As it develops, FINN's approach, with its capacity for mixed-precision quantization and configurable folding, will enable tailored optimization for each layer's bit-width, accuracy, resource usage, and latency.

G. FPGA Implementation with Xilinx Vitis-AI

- 1) Workflow Overview: Figure 8 outlines the workflow used to deploy a neural network on a Xilinx Zynq Ultrascale+MPSoC using the Vitis-AI framework. This approach, distinct from rapid prototyping, is focused on actual embedded deployment. The process begins with training a Float32 model using Keras, followed by exporting it through the Vitis-AI toolkit version 2.0. Deployment on the MPSoC involves four primary activities:
 - Configuring the DPU (Deep Learning Processor Unit) accelerator and generating the FPGA bitstream. This includes selecting the number of DPU cores and their size, which dictates the operations per clock cycle.
 - Generating the application code in C++ to orchestrate model execution using the VART (Vitis AI Runtime).
 - Compiling the model using 8-bit quantization with the Vitis-AI tools.
 - Creating the Board Support Package (BSP) for the ZCU102 board.

Following these steps, the model is executed on the MPSoC, with the DPU on the FPGA handling most of the network operations. However, the CPU may process some layers, particularly when specific functions like the sigmoid activation at the end are not supported by the Vitis-AI quantization.

2) Quantitative Evaluation: Table X presents the evaluation metrics obtained from the validation set using the Vitis-AI workflow, with the initial line providing a baseline from Float32 training with Keras. Following the model's quantization to 8-bit using Vitis-AI, no loss in accuracy was observed, thanks to the toolkit's effective calibration function. The quantized model was subsequently deployed on the Xilinx DPU on the ZCU102 board, where no degradation in performance was noted, suggesting a possible regularization effect.

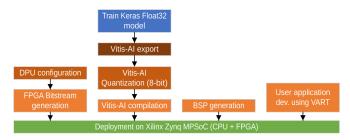


Fig. 8: FPGA workflow from Keras/TensorFlow training to FPGA/CPU inference using Vitis-AI

TABLE X: Evaluation metrics along the Vitis-AI workflow

Model	IoU	Accuracy
Float32 Keras	0.7108	0.9531
Int8 Vitis	0.7156	0.9542
Int8 DPU	0.7263	0.9583

Table XI summarizes the implementation metrics on the Xilinx ZCU102 board, measured on the validation set. The configuration uses three DPU cores, each capable of 4096 operations per clock cycle at 100 MHz. This setup was determined to be the best trade-off for embedded inference, balancing throughput and power consumption for optimal energy efficiency.

TABLE XI: Implementation metrics on the Xilinx ZCU102 with the Vitis-AI workflow

Board	Throughput (FPS)	Power (W)	Energy efficiency (J/image)	Peak mem- ory (MB)
ZCU102	46.9	2.51	53.5	31

Table XII shows the FPGA resource utilization, with DSPs and BRAMs being the primary limiting factors due to their roles in MAC operations and storage of weights and intermediate feature maps, respectively. LUTs, LUTRAMs, and Flip-Flops still have available capacity, providing potential for future increases in the size or number of DPU cores.

TABLE XII: Vitis-AI FPGA resource usage with 3-core DPU on ZCU102 board

FPGA resource	Post-implementation utilization	Available
LUT	133,425 (49%)	274,080
LUTRAM	17,027 (12%)	144,000
Flip-Flop	297,576 (54%)	548,160
BRAM	771 (84%)	912
DSP	2,070 (82%)	2520

3) Qualitative Evaluation: The Vitis-AI workflow is robust, demonstrating significant maturity, particularly with toolchain updates in versions 2.0 and 2.5 that resolved previously encountered bugs. This versatile workflow supports a wide array of neural network layers, and users can incorporate custom IP blocks to introduce new operations. Xilinx provides comprehensive documentation and end-to-end examples through the Vitis-AI Model Zoo. The community surrounding Vitis-AI has grown rapidly, although the learning curve remains steep

due to the complexity of integrating various components such as BSP, Vivado, PetaLinux, and Vitis-AI tools. Additionally, while most components of Vitis AI are open source, some elements, such as the Vitis AI Compiler, remain proprietary, and certain tools within the Xilinx ecosystem require a commercial license.

V. SYNTHESIS

A. Synthesis and Workflow Comparison

Table XIII synthesizes the evaluation and implementation metrics results across the five workflows. As discussed in Section IV, onboard implementation was not achievable for the cuDNN and FINN workflows. Consequently, the implementation results from these workflows are estimates and should be interpreted with caution. Quantization is employed only when the hardware target does not support Float32 operations. The CPU and GPU workflows maintain the neural network's output quality, thus achieving the same accuracy and Intersection over Union (IoU) as their respective baselines. The FINN workflow causes a slight degradation in accuracy and IoU, which is minimal considering the use of low-bit quantization. Conversely, the Vitis-AI workflow marginally improves the evaluation metrics on the validation set due to its quantization and calibration mechanisms, introducing a regularization effect. All workflows are compared at isoaccuracy levels. Nevertheless, there are significant differences in throughput and power consumption across the platforms and workflows. As expected, the CPU exhibits the lowest throughput, resulting in poor energy efficiency. The FPGA workflows, utilizing FINN or Vitis-AI, demonstrate superior energy efficiency. Both FINN and Vitis-AI enable the creation of customizable neural network accelerators, allowing for tailored FPGA resource usage, which in turn affects throughput and power consumption. Additionally, the use of quantization contributes to reduced power consumption. In contrast, GPU workflows and platforms have a considerably higher memory footprint compared to CPU and FPGA workflows and targets, presenting potential challenges in an embedded context.

Table XIV synthesizes the engineering metrics across the five workflows. Overall, TensorFlow and TVM stand out in the comparison. Both are open-source, mature, user-friendly, well-documented, and supported by large, active communities. Close behind, the Vitis-AI workflow exhibits similar positive attributes but is more challenging to use due to its incorporation of proprietary components and a requirement for hardware engineering expertise. Nevertheless, it offers greater customization capabilities than the TVM and TensorFlow workflows. The post-training 8-bit quantization in Vitis-AI, while adding complexity and development time, enhances energy efficiency. The cuDNN workflow is primarily designed for developers of deep neural network frameworks, such as PyTorch and TensorFlow, reflecting its maturity but also its limited suitability for embedded inference. Furthermore, the absence of certain operators, like transposed convolution and nearest neighbor upsampling, necessitates intricate and laborintensive development. At the bottom of our comparison is

TABLE XIII: Synthesis of the evaluation and implementation metrics of the five workflows

Platform	Nvidia GPU		Xilinx Zynq UltraScale+ MPSoC		
Board	Jetson AGX Xavier		ZCU104		ZCU102
Workflow	TensorFlow	cuDNN	TVM (CPU)	FINN (FPGA)	Vitis-AI (FPGA)
Implementation	Yes	No	Yes	No	Yes
Numeric precision	Float32	Float32	Float32	W1A4	Int8
Accuracy change (vs. baseline)	0%	0%	0%	-0.43%	+0.52%
IoU change (vs. baseline)	0	0	0	-0.0271	+0.0155
Throughput (FPS)	74.6	54.3	2.04	127	46.9
Power (W)	14.6	5.61	1.11	5.46	2.51
Energy efficiency (mJ/image)	195	103	543	43.0	53.5
Memory (MB)	2200	1310	78.70	N/A	31

TABLE XIV: Synthesis of the engineering metrics of the five workflows. Metrics are quantified as high, medium, and low.

Platform	Nvidia Jetson AGX GPU		Xilinx Zynq UltraScale+ MPSoC		
Workflow	TensorFlow	cuDNN	TVM (CPU)	FINN (FPGA)	Vitis-AI (FPGA)
Maturity	High	High	High	Low	High
Ease of Use	High	Low	High	Low	Medium
Documentation and Examples	High	Medium	High	Medium	High
Community Support	High	Low	High	Low	High

the FINN workflow. Its current maturity level is low, with identified bugs, and it presents significant usability challenges. The need to develop custom transformations not available in the FINN library further complicates its usage. Although the community is active, it is relatively small compared to the others. Documentation and examples exist but are dispersed across various websites and GitHub repositories, which complicates the comprehension process. Additionally, FINN's lack of support for certain operators, such as transposed convolution, necessitates alterations in the neural network architecture.

B. Limitations and Future Works

The conclusions presented in this paper reflect observations from 2021 to 2023. Nevertheless, the field of neural networks is rapidly evolving, and significant changes in these frameworks are anticipated in the near future. For instance, during the course of our project, we observed maturation in both the FINN and Vitis-AI workflows.

On the GPU front, our research focused on the high-level TensorFlow and the low-level cuDNN workflows. Nvidia's TensorRT, an intermediate, open-source workflow for DNN inference, represents a potential area for future research [23]. Future investigations should also explore quantization to fully leverage the capabilities of Nvidia's Tensor Cores in embedded GPUs, potentially narrowing the energy efficiency gap with FPGAs.

Further research should evaluate the use of more powerful CPUs, such as those based on Intel x86 architectures, with compilers like TVM or Intel nGraph that have shown effectiveness on these processors [14]. Due to time constraints, this study did not explore ASICs for neural network inference, such as Google's Edge TPU or Intel's Movidius VPUs [27], which appear to be promising for embedded applications and warrant future evaluation.

Lastly, the embedded domain poses unique challenges regarding robustness and explainability, aspects not covered in this paper. These topics are currently active research areas

in both academic [21], [34] and industrial spheres [6], [19], deserving attention in future studies.

VI. CONCLUSION

This paper has demonstrated the necessity of adapting advanced neural network architectures to novel datasets within an embedded framework. We introduced a lightweight U-Net that achieves the same accuracy with 16 times fewer parameters and Multiply-Accumulate (MAC) operations, validated on an aerial image segmentation dataset [17]. Furthermore, this study provided an extensive evaluation and comparison of various methods for real-time semantic segmentation of aerial images, employing three contemporary Commercial Off-The-Shelf (COTS) embedded computers across five distinct workflows.

The FPGA target, utilizing Vitis-AI, emerged as the superior choice due to its performance, energy efficiency, and system maturity. However, its implementation necessitates specialized hardware expertise. The ARM CPU target, leveraging TVM, is notable for its user-friendliness and maturity, yet its relatively low energy efficiency and throughput pose significant challenges for embedded system applications. The GPU target, utilizing TensorFlow, is acknowledged for its maturity and ease of use but is more appropriate for rapid prototyping than for actual embedded solutions. Conversely, the GPU target employing cuDNN is better aligned with embedded deployment but suffers from complexity and a lack of support for various neural network layers. Lastly, the FPGA target using FINN shows high potential for energy-constrained applications but necessitates additional development to become a practical option.

VII. ACKNOWLEDGEMENTS

This work was conducted within the SPOC project at the French Institute of Technology (IRT) Saint Exupéry. Funding was provided by the French Research Agency (ANR) and by the industrial partners of the IRT Scientific Cooperation Foundation (FCS).

REFERENCES

- K. Abdelouahab, M. Pelcat, J. Serot, and F. Berry. Accelerating CNN inference on FPGAs: A Survey, May 2018. arXiv:1806.01683 [cs].
- [2] C. Adams, A. Spain, J. Parker, M. Hevert, J. Roach, and D. Cotten. Towards an Integrated GPU Accelerated SoC as a Flight Computer for Small Satellites. In 2019 IEEE Aerospace Conference, pages 1–7, Mar. 2019. ISSN: 1095-323X.
- [3] 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, 11(3):1–23, Dec. 2018.
- [4] L. Brown. Accelerate Machine Learning with the cuDNN Deep Neural Network Library, Sept. 2014.
- [5] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. In 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), pages 578–594, 2018.
- [6] Confiance AI. Un collectif français d'envergure inédite pour concevoir et industrialiser des systèmes à base d'intelligence artificielle de confiance, 2023. Accessed: 2023-05-01.
- [7] I. Dimitrovski, I. Kitanovski, D. Kocev, and N. Simidjievski. Current trends in deep learning for Earth Observation: An open-source benchmark arena for image classification. *ISPRS Journal of Photogrammetry* and Remote Sensing, 197:18–35, Mar. 2023.
- [8] X. Feng, Y. Jiang, X. Yang, M. Du, and X. Li. Computer vision algorithms and hardware implementations: A survey. *Integration*, 69:309–320, Nov. 2019.
- [9] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang. [DL] A Survey of FPGA-based Neural Network Inference Accelerators. ACM Transactions on Reconfigurable Technology and Systems, 12(1):2:1–2:26, Mar. 2019.
- [10] D. Karapetyan, S. Mitrovic Minic, K. T. Malladi, and A. P. Punnen. Satellite downlink scheduling problem: A case study. *Omega*, 53:115–123, June 2015.
- [11] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization, Jan. 2017. arXiv:1412.6980 [cs].
- [12] B. B. Kövari and E. Ebeid. MPDrone: FPGA-based Platform for Intelligent Real-time Autonomous Drone Operations. In 2021 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), pages 71–76, Oct. 2021. ISSN: 2475-8426.
- [13] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015. Number: 7553 Publisher: Nature Publishing Group.
- [14] M. Li, Y. Liu, X. Liu, Q. Sun, X. You, H. Yang, Z. Luan, L. Gan, G. Yang, and D. Qian. The Deep Learning Compiler: A Comprehensive Survey. arXiv:2002.03794 [cs], Aug. 2020. arXiv: 2002.03794.
- [15] Y. Li, Z. Liu, K. Xu, H. Yu, and F. Ren. A GPU-Outperforming FPGA Accelerator Architecture for Binary Convolutional Neural Networks. ACM Journal on Emerging Technologies in Computing Systems, 14(2):18:1–18:16, 2018.
- [16] Z. Lv, T. Liu, J. A. Benediktsson, and N. Falco. Land Cover Change Detection Techniques: Very-high-resolution optical images: A review. *IEEE Geoscience and Remote Sensing Magazine*, 10(1):44–63, Mar. 2022. Conference Name: IEEE Geoscience and Remote Sensing Magazine.
- [17] E. Maggiori, Y. Tarabalka, G. Charpiat, and P. Alliez. Can semantic labeling methods generalize to any city? the inria aerial image labeling benchmark. In 2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS), pages 3226–3229, Fort Worth, TX, July 2017. IEEE.
- [18] S. Mittal. A Survey on optimized implementation of deep learning models on the NVIDIA Jetson platform. *Journal of Systems Architecture*, 97:428–442, Aug. 2019.
- [19] A. Mojsilovic. Introducing AI Explainability 360, Aug. 2019.
- [20] K. Muhammad, S. Khan, M. Elhoseny, S. Hassan Ahmed, and S. Wook Baik. Efficient Fire Detection for Uncertain Surveillance Environment. *IEEE Transactions on Industrial Informatics*, 15(5):3113–3122, May 2019. Conference Name: IEEE Transactions on Industrial Informatics.
- [21] H. Nori, S. Jenkins, P. Koch, and R. Caruana. InterpretML: A Unified Framework for Machine Learning Interpretability, Sept. 2019. arXiv:1909.09223 [cs, stat].

- [22] E. Nurvitadhi, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. Ong Gee Hock, Y. T. Liew, K. Srivatsan, D. Moss, S. Subhaschandra, and G. Boudoukh. Can FPGAs Beat GPUs in Accelerating Next-Generation Deep Neural Networks? In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA '17, pages 5–14, New York, NY, USA, Feb. 2017. Association for Computing Machinery.
- [23] NVIDIA Corporation. Nvidia tensorrt. https://github.com/NVIDIA/ TensorRT, 2023.
- [24] A. P érez, A. Rodríguez, A. Otero, D. G. Arjona, A. Jiménez-Peralo, M. A. Verdugo, and E. De La Torre. Run-Time Reconfigurable MPSoC-Based On-Board Processor for Vision-Based Space Navigation. *IEEE Access*, 8:59891–59905, 2020. Conference Name: IEEE Access.
- [25] A. Pappalardo. Xilinx/brevitas, June 2021. original-date: 2018-07-10T22:37:01Z.
- [26] B. Peccerillo, M. Mannino, A. Mondelli, and S. Bartolini. A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives. *Journal of Systems Architecture*, 129:102561, Aug. 2022.
- [27] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner. Survey of Machine Learning Accelerators. 2020 IEEE High Performance Extreme Computing Conference (HPEC), pages 1–12, Sept. 2020. arXiv: 2009.00993.
- [28] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation, May 2015. arXiv:1505.04597 [cs].
- [29] S. Sharma, C. Beierle, and S. D'Amico. Pose estimation for non-cooperative spacecraft rendezvous using convolutional neural networks. In 2018 IEEE Aerospace Conference, pages 1–12, Mar. 2018.
- [30] N. Tijtgat, W. Van Ranst, B. Volckaert, T. Goedeme, and F. De Turck. Embedded Real-Time Object Detection for a UAV Warning System. In 2017 IEEE International Conference on Computer Vision Workshops (ICCVW), pages 2110–2118, Venice, Oct. 2017. IEEE.
- [31] F. Wang, X. Luo, Q. Wang, and L. Li. Aerial-BiSeNet: A real-time semantic segmentation network for high resolution aerial imagery. Chinese Journal of Aeronautics, 34(9):47–59, Sept. 2021.
- [32] M. Wu, C. Zhang, J. Liu, L. Zhou, and X. Li. Towards Accurate High Resolution Satellite Image Semantic Segmentation. *IEEE Access*, 7:55609–55619, 2019. Conference Name: IEEE Access.
- [33] Y. Xing, J. Weng, Y. Wang, L. Sui, Y. Shan, and Y. Wang. An In-depth Comparison of Compilers for Deep Neural Networks on Hardware. In 2019 IEEE International Conference on Embedded Software and Systems (ICESS), pages 1–8, June 2019.
- [34] F. Xu, H. Uszkoreit, Y. Du, W. Fan, D. Zhao, and J. Zhu. Explainable AI: A Brief Survey on History, Research Areas, Approaches and Challenges. In J. Tang, M.-Y. Kan, D. Zhao, S. Li, and H. Zan, editors, Natural Language Processing and Chinese Computing, Lecture Notes in Computer Science, pages 563–574, Cham, 2019. Springer International Publishing.
- [35] R. Zhao, S. Liu, H.-C. Ng, E. Wang, J. J. Davis, X. Niu, X. Wang, H. Shi, G. A. Constantinides, P. Y. K. Cheung, and W. Luk. Hardware Compilation of Deep Neural Networks: An Overview. In 2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP), pages 1–8, July 2018. ISSN: 2160-052X.