PEFSL: A deployment Pipeline for Embedded Few-Shot Learning on a FPGA SoC

Lucas Grativol, Lubin Gauthier, Mathieu Leonardon, Jérémy Morlier, Antoine Lavrard-Meyer, Guillaume Muller, Fresse, Virginie, Matthieu Arzel

PEFSL: A deployment Pipeline for Embedded Few-Shot Learning on a FPGA SoC

Lucas Grativol*‡, Lubin Gauthier*, Mathieu Léonardon*, Jérémy Morlier*, Antoine Lavrard-Meyer*,
Guillaume Muller‡, Virginie Fresse† and Matthieu Arzel*
*IMT Atlantique, Lab-STICC, UMR CNRS 6285, F-29238 Brest, France
†Hubert Curien Laboratory, Saint-Etienne, France
‡Mines Saint-Etienne, Institut Henri Fayol, Saint-Etienne, France

***Abstract*—This paper tackles the challenges of implementing few-shot learning on embedded systems, specifically FPGA SoCs, a vital approach for adapting to diverse classification tasks, especially when the costs of data acquisition or labeling prove to be prohibitively high. Our contributions encompass the development of an end-to-end open-source pipeline for a few-shot learning platform for object classification on a FPGA SoCs. The pipeline is built on top of the Tensil open-source framework, facilitating the design, training, evaluation, and deployment of DNN backbones tailored for few-shot learning. Additionally, we showcase our work's potential by building and deploying a low-power, low-latency demonstrator trained on the MiniImageNet dataset with a dataflow architecture. The proposed system has a latency of 30 ms while consuming 6.2 W on the PYNQ-Z1 board.**

I. Introduction

For object classification, the conventional approach involves training a neural network using a big labeled dataset. However, such datasets are not always available, usually because the cost of labeling is high [1]. Another, more innovative method is to use a pre-trained network and to specialize it on few labeled examples. This can be performed with transfer learning or fine-tuning [2]. But when the number of labeled examples is really low, the method to be used is called *few-shot learning* [3]. Few-shot learning seeks to leverage the knowledge from deep learning (DL) models to achieve robust classification performance on new tasks, when only a handful of labeled samples per class are available.

One of the primary obstacles to the implementation of few-shot learning on embedded systems is the required computational power induced by the underlying cost of Deep Neural Networks (DNN) models. Careful design of low-complexity DNN adapted to embedded hardware targets is therefore a main concern [4]. Among the potential hardware that can be found in embedded systems, FPGA SoCs (System-On-Chip) have proven to be good candidates for the deployment of DNNs when energy consumption is critical [5], [6] or when low latency is at stake [7]. However, there have been few examples of such FPGA implementations in the literature so far. The challenges to be tackled toward such an implementation are the selection and adaptation of deployment frameworks, the identification and adaptation of an efficient training routine from the literature, and finally the design of a lightweight network that meets the constraints of embedded systems while also performing well for the defined task, few-shot learning on embedded FPGA SoC, with a real-time classification of a video stream.

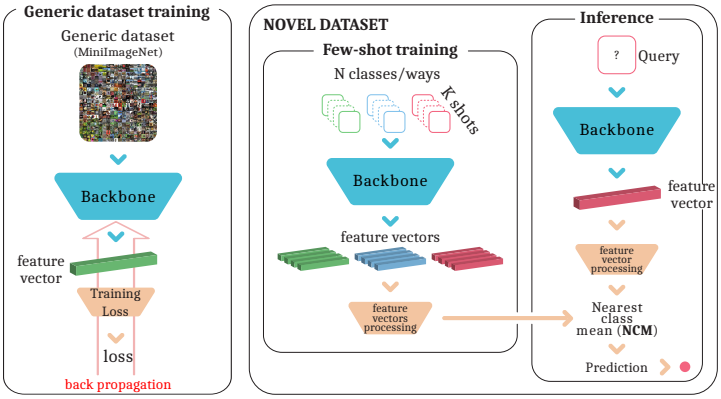

Fig. 1. Our few-shot learning method.

In this paper, we tackle these challenges. Our contributions can be summed up as:

- one of the first few-shot learning platforms for real-time object classification on an FPGA SoC in the literature,
- a full open source pipeline[1], based on the Tensil framework[2], for the design, training, evaluation and deployment of DNN backbones for few-shot learning on FPGA SoCs,
- the demonstration of the potential of this platform on a use case, the design and deployment of a low power and low latency few-shot model on the MiniImageNet dataset on a given hardware architecture.

These contributions aim to pave the way for exciting new applications in fields such as robotics, drones, and autonomous vehicles, where responsiveness, computational power, and energy are critical factors. The entire source code needed to replicate all aspects of this work are open source.

II. Few-Shot Learning

Few-shot learning consists in classifying examples for unseen classes with a very small number of training examples. State-of-the-art methods are based on DL approaches. This may, at first, be counter-intuitive as DL is known to perform

[1]https://github.com/brain-bzh/PEFSL
[2]https://www.tensil.ai/

well when fed with huge databases on which it excels at generalizing.

Our few-shot paradigm is depicted in Fig. 1. The first step (generic dataset training) consists in training a DNN, called backbone, following the method detailed in [3]. This is performed by training a classification network with an additional pretext loss [8]. Few-shot datasets are usually split between the *base* and *validation* dataset, the latter being used to assess the generalization performance of the model. On the contrary to standard classification datasets, in the case of few-shot learning, the classes of the validation set are distinct from those of base set [9], in order to evaluate the generalization performance on new classes. Once trained, the backbone is kept frozen for the subsequent steps, as its only function is to map the input to an high dimension, the feature vectors.

Then, in the next step, the few-shot learning performance is evaluated on a third set of images called the *novel* dataset. This novel dataset consists of thousands of few-shot episodes [10]. In each episode there is a certain number of classes, called *ways*. For each way, there will be a given number of labeled examples called *shots* and some unlabeled ones called *queries*, as depicted in the "Few-shot training" and "Inference" diagrams in the Fig. 1. The performance of the model corresponds to the number of queries that are correctly identified using the few available shots, averaged on the thousands of episodes. The number of shots and ways are set by the benchmarks. As an important distinction in the few-shot learning domain, we aim to solve an inductive [11] problem, when one doesn't have access to the whole set of queries beforehand, and not a transductive [12] one, where one has access to the queries.

Because the DNNs used as backbones are usually complex in terms of memory footprint and computational complexity, the efficiency of these methods in embedded environments remains a challenge. Though, rapid adaptation to new tasks using minimal resources is essential, especially for applications such as real-time object recognition on embedded systems like drones or autonomous robots. Therefore, specific effort has to be made on the design of the backbones.

III. Backbones

A. Architecture

In this experiment, we use ResNets, adapted from [13]. The primary feature of a ResNet is the use of residual blocks, where bypasses between certain layers of the network are added. The main advantage of this network architecture is the ability to train much deeper and more accurate networks than traditional Convolutional Neural Networks (CNN) such as VGGs or AlexNet [14]. This type of network is particularly efficient for our experiment. Indeed, they allow for performance that is very close to the state of the art [3]. Though, they are small networks with relatively few parameters and limited computational complexity.

B. Hyperparameters

Here, we list the main hyperparameters that influence the final system performance and its complexity:

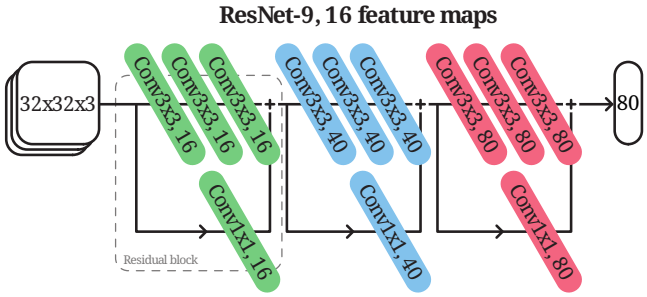

Fig. 2. Structure of a ResNet-9, where initial layers employ 16 output feature maps, and subsequent layers scale their output channels accordingly.

a) Network depth: We choose two ResNet architectures with small depths, ResNet-9 and ResNet-12. A ResNet-9 is simply a ResNet-12 with the last residual block removed. It is expected that the shallower and less computationally intensive ResNet-9 may exhibit lower accuracies for complex tasks. This specific ResNet-9 architecture is depicted in Fig. 2.

b) Training and test image size: The size of training images impacts the amount of computation to be performed, and it also affects performance. Smaller images, like 32×32, contain less information than 100×100 images, but processing them requires fewer operations. As we will see in section V, the joint choice of testing and training image size resolution has a huge impact on the accuracy of the model.

c) Downsampling: Between each residual block, the resolution of the feature maps is reduced. We have two ways to perform this reduction. Either we change the strides of the last convolution in each block from 1 to 2, or we use max pooling, which consists in retaining only the maximum value of groups of values in the feature maps. A stride of 2 or a 2×2 pooling size are equivalent in terms of dimension reduction.

d) Number of feature maps: The backbone is mainly composed of convolution layers. Where the number of filters used on a layer defines the number of feature maps output by that layer. We set the number of filters in the first convolution layer as a hyperparameter, scaling subsequent layers accordingly.

C. Training

We use the MiniImageNet [15] dataset, extracted from ImageNet [16]. It consists of 64 base classes, 16 validation classes, and 20 novel classes. Each class contains 600 images, and the resolution is 84×84. In this paper, we focused on the 5-ways, 1-shot setup. Nevertheless, it has been noticed that performance of a given model and training routines are usually closely correlated across different numbers of shots [3]. The MiniImageNet dataset is specifically designed for few-shot learning. Its value lies in the fact that it contains highly diverse classes that allow for excellent generalization to new classes.

IV. Open Source Pipeline

A. PEFSL pipeline

In order to explore the search space of the previously defined training and network architectures hyperparameters, we devel-

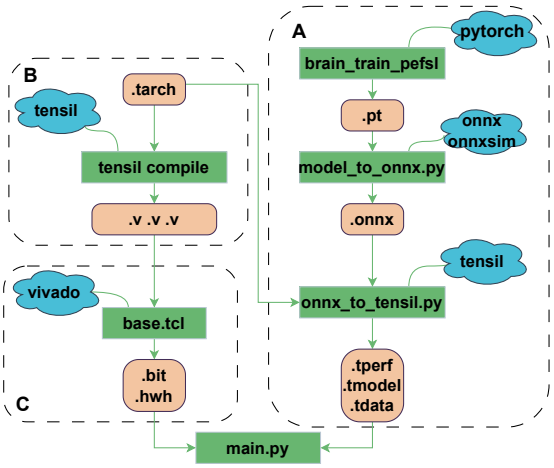

Fig. 3. Modular pipeline for the deployment of a few-shot learning system on an FPGA SoC.

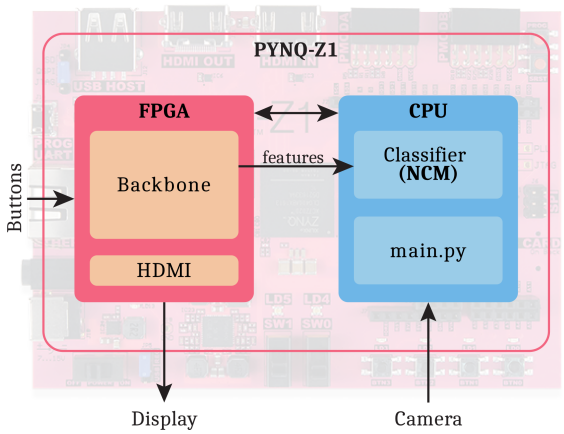

Fig. 4. Schematic of the system.

oped and released PEFSL, a modular pipeline for the training, compilation, hardware synthesis and deployment of a few-shot learning application on an FPGA SoC. It uses several tools that will be detailed hereafter and are depicted in Fig. 3. Part **A** of PEFSL corresponds to the training routine of the backbone, as described in section 1, its conversion in the ONNX format, and its compilation with the Tensil framework. Tensil is an open-source framework for running machine learning models on custom accelerator architectures. The training routine is adapted from [3] in which we added the ResNet-9 and ResNet-12 architectures and their variants. It is using state-of-the-art techniques for training such CNNs on few-shot learning tasks. Then, the pytorch model is translated into an ONNX format. We also use the ONNX simplifier tool that allows for efficient simplification on the ONNX model. Finally, this ONNX model is compiled with the Tensil framework. Provided a description of the underlying architecture (`.tarch` file), that specifies the features of the systolic arrays [17] (number of Processing Elements, data format, memory size). This first three scripts allow for generating automatically the latency of the neural network on the given architecture. Therefore it can be used to perform a design space exploration of the neural network architectures and training techniques, such as in Fig. 5.

Part **B** corresponds to the compilation of the architecture that generates RTL files of the Tensil accelerator IP. These RTL files are used in part **C**, which provides project files to the AMD-Xilinx Vivado tool that generates the bitstream of the PL (Programmable Logic) used in the demonstrator. The produced intermediary files (bitstream and Tensil model) are then used in the main script, which uses the PYNQ driver, that is used for the data transfer between the CPU and the FPGA.

B. Demonstrator

In order to demonstrate how easily this work is applicable in an industrial application context, we created a standalone demonstrator in a compact box. Fig. 4 shows a schematic of our demonstrator, it consists of the PYNQ-Z1 board, a 800x540p HDMI screen, a 160x120p camera, and a 10,000mAh battery. It has a 5.75-hour battery life during inference. The demonstration includes on screen indicators for a better user experience. With the entire system and indicators, we achieve an average of 16 FPS during inference. The network is a ResNet-9 with 16 feature maps. The inference runs in the FPGA at 125 MHz, it is implemented using a 16-bit fixed-point format with 8 bits designated for the integer part. The entire system, encompassing the SoC, camera, and screen, operates with a power consumption of 6.2W. On the programmable logic are implemented a Tensil hardware accelerator and an HDMI Xilinx IP which are using most of the FPGA resources. Then, all the software including pre-processing, post-processing, and image classification is executed on the CPU. The demonstrator includes interfaces to camera and buttons to control a live demo.

For the hardware implementation we use the base architecture proposed by Tensil for the PYNQ-Z1 board, increasing only the size of the systolic array from 8×8 to 12×12, which corresponds to the highest possible value to fit in the FPGA alongside the HDMI controller. In the current version of the pipeline, the NCM classifier is implemented on the CPU side, in a future version we intend to move it to the FPGA.

V. Results

A. Design Space Exploration

The training results are presented in Fig. 5. The hyperparameters search space defined in section III was exhaustively explored. We compiled each network with Tensil to obtain the number of cycles taken by the network's inference. In order to get a smooth video stream (greater than 10 FPS), it is necessary to work with 32×32 images. Therefore, we show the results for this resolution alongside the 84×84 resolution that is classically chosen for experiments on the MiniImageNet dataset, in a 5-ways, 1-shot setup. The first takeaway is that for the 32×32 resolution, ResNets-9 (empty marks) exhibit higher accuracies than the ResNets-12 (full marks), despite their lower number of layers and parameters.

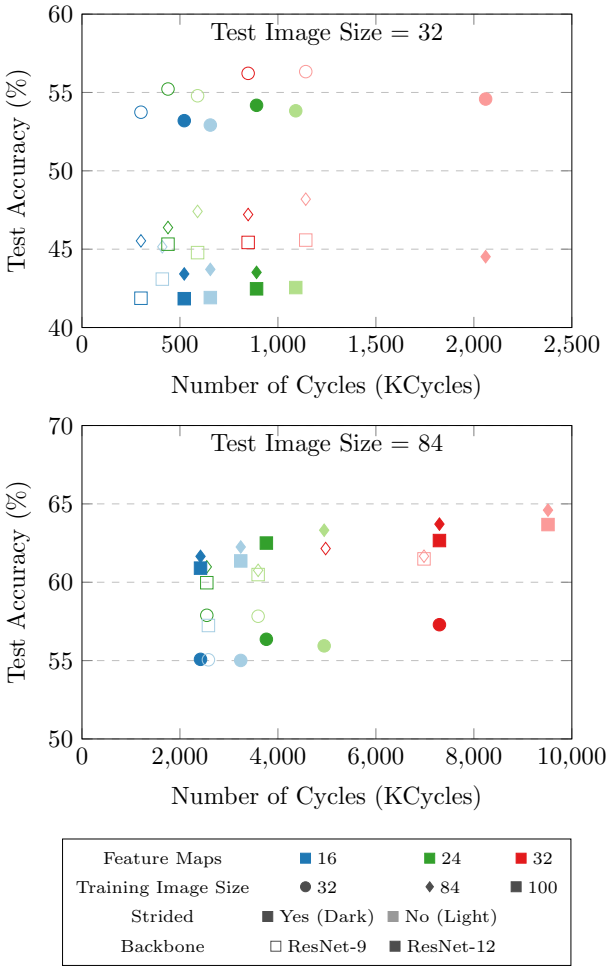

Fig. 5. Accuracy and Latency Trade-off: Graphs depict tests on 32×32 (top) and 84×84 (bottom) images. Different feature maps configurations are denoted by unique colors, while distinct training image sizes are represented by different shapes. We also investigate the impact of strided architectures, differentiated by dark and light colors. Additionally, we vary the backbone architecture from ResNet-9, with empty forms, and ResNet-12, filled forms.

We hypothesized that, with 32×32 images on a ResNet-12, the dimension of feature maps in the last layer is too small and hardly exploitable by the downstream NCM.

The second takeaway is that for a target test resolution of 32×32, the training image sizes should be the same, 32×32 (circles). Indeed, the networks trained on larger images 84×84 and 100×100 are far less accurate. This could be counter-intuitive as training with images resized on 32×32 means that some information is lost in the dataset. It is possible that an even better accuracy could be obtained with clever data augmentation or a better generalization error metric [18].

Using convolutions with a stride of 2 in the network allows for a reduction in the number of operations to be executed when compared to using max pooling layers to reduce the dimensions of intermediate representations in the network. This is denoted as *strided* in the Fig 5. We verify that the latency is reduced in this case, but also that the accuracy is not impacted by this change, if not increased. Finally, the number of feature maps of the first layer, used as a way to scale the width of the network, allows for a trade-off between latency and accuracy.

In summary, for our specific application, the optimal trade-off lies in the top-left corner, where we can identify configurations with acceptable accuracy and the lowest latency. Consequently, we have selected the strided ResNet-9, trained with 32×32 images and 16 feature maps, utilizing 32×32 images during inference, empty blue circle on the first graph of Fig. 5.

TABLE I
CIFAR-10 INFERENCE ON Z7020 FPGA

Work	Prec. [bits]	LUT	BRAM [36 kb]	FF	DSP	Latency [ms]	Acc. [%]
[21] hls4ml	8-12	28544	42	49215	4	27.3	87
[21] FINN	1	24502	100	34354	0	1.5	87
[22]	1-2	23436	135	-	53	1.1	86
[23]	16	15200	523	41	167	109	-
Ours	16	15667	59	9819	159	35.9	92

B. Comparison with other hardware implementations

We set the array size of our systolic array to 12, which corresponds to the maximum possible array size for our setup. The FPGA frequency has been set to 125MHz. Under this configuration, the latency of the backbone inference is 30ms. Few articles have specifically addressed few-shot learning on FPGAs or in embedded systems. An example of few-shot pest recognition on an FPGA has been proposed in [19], reaching 2 frames per second on a PYNQ-Z1. To demonstrate that the hardware resources and latency obtained using Tensil's framework, based on the computational complexity of our backbone, are within the standards of the literature, we conducted a benchmark and present the results in Table I. We decided to compare with implementations of DNNs proposed for classification on the CIFAR-10 dataset [20]. Indeed, these are images with a resolution of 32×32 pixels, for which the backbone we have chosen (ResNet-9 with 16 feature maps) is highly adaptable, provided that we add a downstream linear layer. We restricted our search to implementations on the same chip as ours, the Zynq-7020 (z7020). From Table I shows that Tensil's implementation offers comparable latency and accuracy for equivalent resources, validating our backbone. It is important to notice each work implements a different DNN. For this benchmark, we use array size of 12 at 50 MHz.

VI. CONCLUSION

In this paper, we propose the first implementation of inductive few-shot learning system on an FPGA SoC, allowing for fast inference and low power consumption. We propose PEFSL, a fully open-source implementation pipeline that allows for designing a neural network architecture, training and deploying it on an embedded system. Our implementation achieves 54% accuracy on the MiniImageNet dataset for the 32×32 resolution in the 1-shot, 5-ways scenario, with a 30ms latency on the PYNQ-Z1 board.

REFERENCES

[1] T. Fredriksson, D. I. Mattos, J. Bosch, and H. H. Olsson, "Data labeling: An empirical investigation into industrial challenges and mitigation strategies," in *International Conference on Product-Focused Software Process Improvement*. Springer, 2020, pp. 202–216.

[2] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big data*, vol. 3, no. 1, pp. 1–40, 2016.

[3] Y. Bendou, Y. Hu, R. Lafargue, G. Lioi, B. Pasdeloup, S. Pateux, and V. Gripon, "Easy—ensemble augmented-shot-y-shaped learning: State-of-the-art few-shot classification with simple components," *Journal of Imaging*, vol. 8, no. 7, p. 179, 2022.

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

[5] 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.

[6] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey and benchmarking of machine learning accelerators," in *2019 IEEE high performance extreme computing conference (HPEC)*. IEEE, 2019, pp. 1–9.

[7] J. Zhang, L. Cheng, C. Li, Y. Li, G. He, N. Xu, and Y. Lian, "A low-latency fpga implementation for real-time object detection," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2021, pp. 1–5.

[8] S. Gidaris, A. Bursuc, N. Komodakis, P. Pérez, and M. Cord, "Boosting few-shot visual learning with self-supervision," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 8059–8068.

[9] P. Mangla, N. Kumari, A. Sinha, M. Singh, B. Krishnamurthy, and V. N. Balasubramanian, "Charting the right manifold: Manifold mixup for few-shot learning," in *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, 2020, pp. 2218–2227.

[10] S. Laenen and L. Bertinetto, "On episodes, prototypical networks, and few-shot learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 24 581–24 592, 2021.

[11] T. Scott, K. Ridgeway, and M. C. Mozer, "Adapted deep embeddings: A synthesis of methods for k-shot inductive transfer learning," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[12] Y. Liu, J. Lee, M. Park, S. Kim, E. Yang, S. J. Hwang, and Y. Yang, "Learning to propagate labels: Transductive propagation network for few-shot learning," *arXiv preprint arXiv:1805.10002*, 2018.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[14] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: analysis, applications, and prospects," *IEEE transactions on neural networks and learning systems*, 2021.

[15] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra *et al.*, "Matching networks for one shot learning," *Advances in neural information processing systems*, vol. 29, 2016.

[16] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[17] R. Xu, S. Ma, Y. Guo, and D. Li, "A survey of design and optimization for systolic array based dnn accelerators," *ACM Computing Surveys*, 2023.

[18] Y. Bendou, V. Gripon, B. Pasdeloup, G. Lioi, L. Mauch, S. Uhlich, F. Cardinaux, G. B. Hacene, and J. A. Garcia, "A statistical model for predicting generalization in few-shot classification," in *2023 31st European Signal Processing Conference (EUSIPCO)*. IEEE, 2023, pp. 1260–1264.

[19] Y. Li and J. Yang, "Few-shot cotton pest recognition and terminal realization," *Computers and Electronics in Agriculture*, vol. 169, p. 105240, 2020.

[20] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[21] H. Borras, G. Di Guglielmo, J. Duarte, N. Ghielmetti, B. Hawks, S. Hauck, S.-C. Hsu, R. Kastner, J. Liang, A. Meza *et al.*, "Open-source fpga-ml codesign for the mlperf tiny benchmark," *arXiv preprint arXiv:2206.11791*, 2022.

[22] L. Yang, Z. He, and D. Fan, "A fully onchip binarized convolutional neural network fpga impelmentation with accurate inference," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2018, pp. 1–6.

[23] H. Kim and K. K. Choi, "A reconfigurable cnn-based accelerator design for fast and energy-efficient object detection system on mobile fpga," *IEEE Access*, 2023.