Challenges in Energy-Efficient Deep Neural Network Training with FPGA

Yudong Tao[1], Rui Ma[1], Mei-Ling Shyu[1], Shu-Ching Chen[2]
[1]Department of Electrical and Computer Engineering
University of Miami, Coral Gables, FL, USA
[2]School of Computing and Information Sciences
Florida International University, Miami, FL, USA
{yxt128, rxm1351, shyu}@miami.edu, chens@cs.fiu.edu

Abstract

In recent years, it is highly demanding to deploy Deep Neural Networks (DNNs) on edge devices, such as mobile phones, drones, robotics, and wearable devices, to process visual data collected by the cameras embedded in these systems. In addition to the model inference, training DNNs locally can benefit model customization and data privacy protection. Since many edge systems are powered by batteries or have limited energy budgets, Field-Programmable Gate Array (FPGA) is commonly used as the primary processing engine to satisfy both demands in performance and energy-efficiency. Although many recent research papers have been published on the topic of DNN inference with FPGAs, training a DNN with FPGAs has not been well exploited by the community. This paper summarizes the current status of adopting FPGA for DNN computation and identifies the main challenges in deploying DNN training on FPGAs. Moreover, a performance metric and evaluation workflow are proposed to compare the FPGA-based systems for DNN training in terms of (1) usage of on-chip resources, (2) training efficiency, (3) energy efficiency, and (4) model performance for specific computer vision tasks.

1. Introduction

As an emerging technique in the field of computer vision, the deep neural network (DNN) has achieved superior performance in various applications. For example, the convolutional neural network (CNN) has proven to be an effective approach to recognize abstract and high-level concepts from unstructured visual data, such as images and videos. ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [9] has witnessed the emergence of numerous milestone image classification models, including VGGNet [39], ResNet [18], etc. Apart from image classification, CNN can also achieve state-of-the-art performance on many other computer vision tasks, such as object detection [36], segmentation [17], and action recognition [30]. Furthermore, other DNN architectures, such as the generative adversarial network (GAN) [13] and graph convolutional neural network (GCNN) [48], have been recently proposed and applied to a broader range of critical computer vision applications, including image generation, image denoising, cloud point segmentation, etc.

Besides the achievements obtained by DNN, it can also be observed that as the model performance improves, the model size also becomes larger and larger. Given sufficient amounts of training samples, deeper and wider DNN models usually perform better than shallower and thinner ones [32]. However, the performance of DNN does not scale linearly with the model size. For example, AmoebaNet-A [34], an extremely-large-scale CNN model with 469 million parameters, is roughly 275 times the size of ResNet-101 [18]. Compared with the explosive model size, the performance improvement of AmoebaNet-A is not that significant: it is only 19.0% better than ResNet-101 in terms of the top-1 error rate on the ImageNet dataset (i.e., reduced the error rate from 19.87% to 16.1%). More importantly, such a large model cannot be deployed on edge devices due to limited resources.

Low power electronics, such as mobile devices, unmanned aerial vehicles (UAVs), and Internet of Things (IoT) sensors, are electronic devices that are designed to operate under a limited power capacity. Recently, smartphones have become the primary platforms of modern computer vision technologies, including face unlock, text recognition, and many others. However, the "most desired" feature of a computer vision application on smartphones is not the performance of the corresponding computer vision algorithms, but how many computational resources it consumes [1]. Therefore, it is demanding to build energy-efficient DNNs that can satisfy the energy budget of edge devices. Moreover, due to the benefits in preserving data privacy, learning in the local environment, such as federated learning [24], has been proposed and attracted much attention in recent years. It is also desired to provide flexibility to users to customize their local recognition applications. To enable such technology at edge, it is necessary to allow the edge devices to train DNNs in an energy-efficient manner. Moreover, the global market size of computer vision is estimated to increase from USD 11.9 billion in 2019 to USD 17.4 billion by 2024, with a compound annual growth rate (CAGR) of 7.8%; whereas the market share of computer vision technology in edge devices is growing exponentially [29]. As a result, training DNNs on edge devices can have a significant impact on the development of low power computer vision technology.

Furthermore, the process of developing a new deep learning model is always energy-intensive. A recent study conducted by Strubell et al. [41] reveals that the estimated carbon emission from training a transformer model [45] with the neural architecture search using GPU can be five times as much as the carbon emission of a car in its whole lifetime. Even though the amount of CO_2 produced from deep learning is still negligible compared with the total carbon emissions from the whole planet, low power solutions are still in urgent need in order to slow down the increasing energy consumption in deep learning.

To improve the efficiency of DNN computation, efficient neural network architectures such as SqueezeNet [21] and ShuffleNet [54] were proposed. These designs could reduce the amounts of parameters and operations without significantly degrading the model performance. Meanwhile, model compression and quantization are introduced to effectively reduce the model size and improve efficiency. Model compression methods [2, 19] aim to prune the redundant structures in the neural networks that have no or minimal impact on the model performance to avoid wastes of memory and computing power. On the other hand, model quantization [15] could reduce the number of bits used in parameters and results of the DNNs so that the computation can become more efficient. DNNs optimized by the aforementioned techniques can achieve comparable performance with significantly smaller memory usage and better computing efficiency. However, many model compression and quantization methods cannot be directly applied during the DNN training stage. Otherwise, the model performance will significantly degrade. Modified optimization methods including stochastic weight averaging [50], low-precision stochastic variance-reduced gradient [37], etc. were recently proposed to train DNNs with low-precision floating numbers.

Despite the efficient algorithms and methods of model compression and quantization, hardware platforms, such as GPUs, might not be flexible enough to support all sorts of optimizations. For example, most of the commercial GPUs support only full-precision floating-point operations. Using fixed-point or even mixed-precision numbers for calculation is not fully supported in most of GPUs. Meanwhile, random network compression might not benefit the computation efficiency on GPUs since the computation is highly parallelized, and the overhead of memory alignment is higher than the gains obtained from model compression. Hence, a hardware-software co-design is required to accelerate the DNN computation at the edge, and the flexibility of hardware platforms thus becomes a key feature. Field-Programmable Gate Arrays (FPGAs) thus become a good candidate, providing good energy efficiency and flexibility to configure the hardware. The advantages and disadvantages of FPGA compared to GPUs will be further discussed in details in Section 2.

The rest of the paper is organized as follows. Section 2 compares various hardware platforms for DNN computation and introduces several existing techniques to accelerate DNN computation on FPGAs. After that, the key challenges in enabling energy-efficient DNN training on FPGAs are discussed in Section 3. Section 4 then proposes a performance metric to evaluate DNN training on the FPGA-based system and a corresponding workflow to benchmark the solutions. In the end, Section 5 concludes the contribution of this paper.

2. Hardware Platforms for DNN Computation

In this section, various hardware platforms for DNN computations are introduced and compared to illustrate the advantages and disadvantages of various platforms. Meanwhile, specific techniques to accelerate DNN computation with FPGAs are briefly discussed and summarized. The energy efficiency of FPGAs for DNN training and inference is also demonstrated.

2.1. Comparing CPU, GPU, and FPGA for DNN Computations

As the accuracy of deep neural networks becomes higher and higher, the model size grows larger to achieve a better representation capability. Thus, both model training and inference of DNNs require the computation accelerators to provide sufficient computation power.

Graphics Processing Units (GPUs) are the most commonly used accelerators for DNN computations. Compared to Central Processing Units (CPUs) that employ a few high-performance floating-point computing cores and optimize the latency of executing instructions, GPUs distribute the tasks among a large number of floating-point computing cores and allow massive parallelism to maximize the throughput. Since the intra-layer and inter-data computations for DNNs are independent, DNN computations can be easily parallelized, and thus GPUs are suitable for accelerating its computations. Moreover, GPUs are well-suited in performing dense floating-point matrix multiplication (GEMM) operations, which are widely used in mainstream DNNs [31].

GPUs processing can achieve the highest throughput for DNN computations. However, it is not energy-efficient since GPUs implement complex control modules to enable the computation pipeline for a general purpose and only have a limited flexibility to handle network sparsity and data types (i.e., most GPUs support only full-precision floating-point operations and some support half-precision floating-point and 8-bit fixed-point operations) [31]. Meanwhile, the latency of GPUs is longer, which is critical to streaming data processing and algorithms with inter-data dependency.

Field-Programmable Gate Arrays (FPGAs), on the other hand, have the potential to address these issues. FPGAs allow the integrated circuit reconfiguration and provide the flexibility to implement wide ranges of operations and instructions. Modern FPGAs contain many different components and on-chip resources, including flip-flops, look-up tables, arithmetic-logic units, communication cores, block RAMs, etc. [26]. In addition, FPGAs do not rely on the Von-Neumann architecture, which can potentially alleviate the bottlenecks in external memory access. Therefore, DNNs on FPGAs can achieve a much better energy efficiency (GOPs/Watts) than GPUs [6].

Moreover, FPGAs can be configured to directly access peripheral hardware components such as sensors or input data sources, which can offer very high bandwidth and much lower latency. On the other hand, the communication between GPUs and hardware components is less efficient. That is, standard buses (USB or PCIe) need to be employed in order to access the hardware, and a host system (e.g., CPU) is also required. As a result, the latency of GPUs is much higher than that of FPGAs. The flexibility of FPGAs also enables an easy deployment of various model compression and quantization methods. For example, Binarized Neural Network (BNN) [20], a recently proposed neural network that achieved nearly state-of-the-art results on multiple benchmark datasets, uses only a 1-bit data type for all weights and activations at run time. Therefore, BNN is well suited to be deployed on FPGAs.

Although FPGAs can offer better energy efficiency, connectivity, and flexibility, one major challenge of using FPGAs is the engineering effort in development. Unlike GPU development that requires only software engineering skills, the development of FPGAs requires hardware configuration skills as well. Moreover, FPGAs do not have as many pre-built packages or libraries as GPUs for DNN computations. Though several existing libraries, such as PYNQ[1], were released to facilitate the high-level implementation of DNN inference on FPGAs, training DNNs on FPGAs still remains a major challenge. Consequently, using FPGAs for DNN computations can be much harder than using GPUs for many deep learning researchers and engineers.

[1]`https://github.com/Xilinx/PYNQ`

Table 1. Performance comparison between CPU, GPU, and FPGA for DNN computations

	CPU	**GPU**	**FPGA**
Throughput	Lowest	Highest	High
Latency	Highest	Medium	Lowest
Power	Medium	Highest	Lowest
Energy Efficiency	Worst	Medium	Best
Device Size	Small	Large	Small
Development	Easiest	Easy	Hard
Library Support	Sufficient	Sufficient	Limited
Flexibility	Limited	Limited	Flexible

Table 1 summarizes the differences among CPU, GPU and FPGA for DNN computations [6, 26, 31]. In short, GPU remains the superior hardware for high-throughput implementations of DNNs; while FPGAs have a great potential in applications where power consumption, power efficiency, and/or latency are of concern. However, in order to make FPGAs more broadly used for DNN acceleration, open-source libraries that provide a high-level abstraction of the hardware programming and alleviate the requirements in hardware configuration skills and knowledge for DNN inference and training. In addition to the homogeneous computing environment (i.e., processing DNNs using a single type of devices), the heterogeneous computing environment can potentially further improve the performance and provide additional flexibility to balance the throughput and power-efficiency [44]. FPGA, GPU, and CPU can incorporate with each other to achieve a better system performance. However, implementing DNN inference and training on such a heterogeneous environment brings additional levels of complexity. To maximize the utility of various hardware platforms, building a simplified and efficient design flow that enables an easier use of FPGAs for DNNs is thus the most important problem to solve.

2.2. DNN Acceleration on FPGAs

Both software and hardware acceleration techniques have been studied for FPGAs-based deep learning. For software acceleration, the main idea is to reduce the computation or bandwidth requirements of deep learning models as much as possible while keeping the accuracy. Current methods toward software acceleration include network optimization, data quantization, and weight reduction [14].

Network optimization aims at simplifying the calculations in deep learning models without losing too much performance, thus reducing the bandwidth requirements of FPGAs. For example, a CNN accelerator design with uniform loop unroll factors across different convolutional layers was proposed, which achieves a performance of 61.62 GFLOPS [51]. In [42], a systematic design space exploration methodology was proposed to maximize the throughput of an OpenCL-based FPGA accelerator of two large-scale CNNs: AlexNet and VGG, achieving a peak performance of 136.5 GOPS for the convolution operation and 117.8 GOPS for the entire VGG network. Another framework that employs a fusion architecture and heterogeneous algorithms to accelerate CNNs on FPGAs was proposed [49]. Zhang et al. proposed a two-step optimization strategy, namely reverse-pruning and peak-pruning, which successfully reduced the size of AlexNet by a factor of 13x without the accuracy loss on a Xilinx Zynq ZCU104 FPGA accelerator [53].

Data quantization is another commonly used model compression approach for deep learning on FPGAs, where the weights and activations in a typical neural network represented by floating-point numbers are replaced by the fixed-point representations with fewer bits. By doing so, the storage space of the deep learning model will be greatly reduced, thereby reducing the computational cost as well as energy consumption. For example, Qiu et al. compared the results from multiple data quantization approaches with different data types, including 48-bit fixed-point, 16-bit fixed-point, and 32-bit floating-point, and presented that the proposed 8/4-bit dynamic-precision quantization only exhibited a 0.4% accuracy loss on the VGG16 model [33]. In [55], the low precision binarized neural network was implemented on a low-cost FPGA development board, ZedBoard, and it outperformed all existing CPU, GPU, and FPGA-based baseline accelerators.

The fundamental idea of weight reduction approaches is to apply low-rank approximation on the model weight in order to reduce the sizes of the weight matrix, thereby reducing the total number of operations. For example, Faraone et al. proposed the reconfigurable constant coefficient multipliers (RCCMs) circuits that multiply the input values by a restricted choice of coefficients using only adders, subtractors, bit shifts, and multiplexers (MUXes) [11]. The proposed design on Xilinx KU115 FPGA achieved 50% resource savings over traditional 8-bit quantized networks.

However, many of the aforementioned techniques cannot be adopted during model training since they might significantly reduce the final model performance. Efficiently training DNNs on FPGAs needs to incorporate low-precision optimization techniques and advanced data structures for the weight representation to achieve state-of-the-art performance without violating the constraint of energy efficiency. To mitigate the loss in accuracy, a low-precision stochastic variance-reduced gradient optimization method that can train the DNN with low-precision weights was proposed recently [37]. The stochastic weight averaging optimization method was proposed afterward to further improve the performance of DNN training with low-precision weights, and it achieved comparable performance compared to using full-precision floating-point numbers [50]. Orthogonal to the optimization methods, dynamic fixed-point was proposed to avoid performance degradation, as well as preserving the small model size [8]. Such dynamic fixed-point numbers are not directly supported by any GPUs or general-purpose processors. However, FPGAs are the most suitable hardware platforms to incorporate such kind of techniques and train DNNs in an energy-efficient manner.

3. Challenges in DNN Training with FPGAs

Current DNN models focus on a static and offline training mechanism, where the training data is prepared in advance. However, it is demanding to train DNNs dynamically and to adapt the models to the local environment. Also, when data privacy is of concern, i.e., the users do not want to share their personal and sensitive information with the AI companies, distributed model training, such as federated learning [24], is more desirable. In both scenarios, training DNN models locally on edge devices is necessary, and thus efficient DNN training is an important research problem to be delivered. Since the low power nature of the edge devices, FPGAs become a good candidate for the primary processor for DNN training.

Although many researchers have focused on DNN inference on FPGAs, very few research papers have explored DNN training on FPGAs, or how to optimize the architecture design on FPGAs for DNN training. Moreover, to find the optimal solution to a specific application on the FPGA platform, both DNN architecture and its FPGA implementation need to be determined with the constraints of limited on-chip resources. Significant research efforts should be invested to achieve the convenient deployment of DNNs on FPGAs for training. The main challenges of implementing DNN training on FPGAs are as follows.

1. When DNNs are trained, computing the gradients of network parameters depends on both the inputs of the current layer and the gradients of its following layer(s). The length and heterogeneity of the data dependency paths of different layers make it difficult to design a FPGA system that could effectively pipeline the processes and avoid external memory accesses.

2. The knowledge required for the DNN design and FPGA design is different. While designing an optimized CNN training framework requires in-depth knowledge in computer vision, deep learning, and optimization, deploying DNNs on FPGAs requires knowledge about logic circuit design and HDL languages.

As shown in Figure 1, the data dependency of an L-layer linear DNN becomes more complicated when the Backward Propagation (BP) is involved. BP requires the intermediate

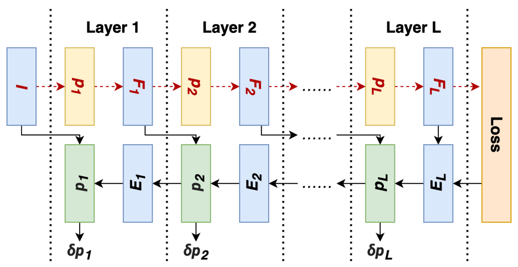

Figure 1. Data dependency of an L-layer linear DNN including both Forward Propagation (FP) and Backward Propagation (BP). Paths of FP are annotated as red dashed arrows, and paths of BP are annotated as black solid arrows. Yellow boxes (at the top-left of each layer) denote the computations in FP, green boxes (at the bottom-left of each layer) denote the computations in BP, blue boxes (on the right of each layer) denote data, and the orange box (on the right of the figure) denotes the loss computation. I is the input data, p_i is the set of parameters of each layer, F_i is the set of output features of each layer, and E_i and δp_i are the partial derivatives of the loss with respect to F_i and p_i, respectively.

outputs of each layer $F_i, i = 1, 2, \ldots, L$ to be reused to compute the gradients of the parameters, δp_i, which are dependent on the partial derivatives of loss with respect to the intermediate outputs of the following layers, i.e., $E_j, j > i$. Such data dependency raises a great challenge in the memory management and data reusing, which are critical to the computing efficiency and power efficiency of DNN training. While reusing the data locally can optimize the performance, more on-chip resources will be consumed. The optimal solution to avoid external memory access varies for different network architectures and computing platforms, i.e., the performance of training DNNs on FPGAs deeply depends on the architecture designs of both the neural networks and hardware.

To address this challenge, several FPGA-based DNN training accelerators have been proposed recently. FPDeep architecture [12] employs the layer partition and mapping strategies and incorporates fine-grained operation pipelining to maximize resource sharing. CNN training with FPDeep can achieve at most 6.36 times gain in energy efficiency. However, such a architecture limits the DNN to be linear, i.e., each layer has at most one preceding layer and one succeeding layer. Therefore, many networks with skipped connections (e.g., ResNet [18], etc.) and other complex structures cannot be trained with the FPDeep architecture. In [46], the data router architecture was proposed to feed data and weights to a multiply-and-accumulate (MAC) array to enable more flexible computation acceleration for CNN training. The proposed architecture can achieve up to 479 GOPS throughput but worse power efficiency for a large batch size. The limited memory bandwidth on the FPGA limits the utilization rate of the FPGA and thus degrades its power-efficiency. Based on the aforementioned state-of-the-art FPGA architecture designs for DNN training, it can be observed that the flexibility of FPGAs cannot be well utilized, and the energy efficiency of CNN training on FPGAs might not be achieved using an inappropriate architecture design.

Furthermore, to the best of our knowledge, all the existing frameworks mainly consider CNNs while other network structures like recurrent neural networks (RNN), graph neural networks (GNN), etc. are not supported. However, a mixed-use of CNNs, RNNs, and GNNs is critical to applications such as video analysis, which increases the complexity of hardware design for network training. Therefore, developing toolkits that can facilitate the design workflow and shorten the design cycle of network training becomes the most important problem to be addressed. Given the wide variety of factors to be considered during design, including network architectures, hardware constraints, performance requirements, etc., it is impossible to have an one-for-all design to train various DNNs on FPGAs. The complexity of this problem makes the manual design process very time-consuming, even for a seasoned FPGA engineer. Consequently, an automated design workflow from the DNN architecture to the hardware design is necessary to enable efficient and effective DNN training for FPGAs. Furthermore, such an automated FPGA design workflow can decouple the needs in the knowledge of the DNN training framework design and hardware design. If an effective automated design workflow is available, researches and engineers can develop DNNs for specific applications without the need to possess deep knowledge about the hardware design.

An automated design for FPGAs is not a new concept. Both Xilinx and Altera, two largest FPGA manufacturers, provide a series of tools to facilitate the design process, including automated place and route, automated design optimization, etc., along with the high-level synthesis (HLS), i.e., converting a high-level programming language, such as C and C++, to a hardware description language (Verilog or VHDL) [7]. However, such a high-level synthesis can barely work for deploying DNNs, especially for training DNNs, since a straightforward conversion from high-level instructions to logic circuit design consumes too many on-chip resources, which is beyond the capability of current commercial FPGA devices. Meanwhile, the general HLS cannot apply model compression and other DNN-specific optimization techniques, which increases the needs in on-chip resources. There exist some HLS software designed for DNN inference, and the model compression and quantization are incorporated to reduce the resource usage. The solutions can mainly be divided into two types, namely processing-engine architecture [38, 52] and grouped-operation architecture [16]. The processing-engine architecture builds an array of general-purpose processors, such as MAC, which optimizes the resource usage and thus can deploy larger models with a larger batch size. The grouped-operation architecture, on the other hand, implements the operations in DNNs at a higher-level, such as a convolution operation processor, which optimizes energy efficiency. However, for training DNNs, the processing-engine architecture which can easily reach the memory access bottleneck due to less memory reuse has been implemented and hit the limits of memory bandwidth [46]; while the grouped-operation architecture uses too many on-chip resources [12]. Therefore, further optimizations are required to realize an effective automated design for efficient DNN training.

It has been witnessed in recent years that Neural Architecture Search (NAS) becomes an effective approach for automated architecture design of DNNs [4, 5, 35, 43, 56]. Furthermore, the hardware efficiency has been considered in some NAS frameworks which optimize the DNN architecture with the consideration of computation efficiency [4, 47]. Feedback from hardware is taken into account for searching the optimal architecture, where the efficiency is considered as a constraint. Following the same strategy, FPGA resources can be used as a constraint in the NAS framework, so the hardware design and the DNN architecture can be optimized simultaneously [16, 22]. Therefore, it is viable to use NAS to optimize the hardware design for DNN training as well. When the hardware design is considered, the type of processors in the array and the operation groups to form become the hyper-parameters of the overall design and thus enlarge the search space. Therefore, how to effectively retrieve the solution should be addressed in the future research.

4. Energy-Efficient DNN Training Benchmark

To address the problem of energy-efficient DNN training with FPGAs, a hardware-software co-design is required. The DNN architectures should be designed with the awareness of the hardware constraints of FPGAs and target performance, and the hardware design needs to be optimized based on the DNN architecture. To evaluate the co-design solution, both DNN-related and hardware-related performance metrics, including on-chip resource usage, training efficiency, energy efficiency, and task-specific model performance, need to be considered for a comprehensive evaluation that reflects the overall design quality. In this section, a performance metric and the evaluation workflow are discussed based on three identified computer vision tasks for the energy-efficient DNN training benchmark.

4.1. Identified Computer Vision Tasks

With the rapid development of computer vision, various computer vision tasks were proposed and solved every year. In this energy-efficient DNN training benchmark, three common computer vision tasks are selected to reflect different potential scenarios that require DNNs and model training. Table 2 summarizes the identified tasks: image classification, video classification, and object detection, as well as their pre-training datasets, fine-tuning and test datasets, the task-specific performance metrics, and the example state-of-the-art DNN architectures for the tasks. The pre-training dataset is used to train the DNN models before deploying to the local device, the fine-tuning dataset is used to train the models on the local FPGA, and the test dataset is used for evaluating the performance after fine-tuning.

Among these tasks, image classification whose objective is to classify an image into a specific category based on its visual content is regarded as the most basic one. There are a large number of datasets for image classification, such as MNIST [27], CIFAR [25], ImageNet [9], and Food-101 [3]. In this benchmark, ImageNet is used to pre-train the model, while the data in the Food-101 datset will be used to fine-tune the model. Top-k accuracy is usually chosen as the evaluation metric for most image classification tasks, which can be defined as follows. Given an image, the image classification model produces a probability of the image belonging to each category. The image is regarded as correctly classified if the true label is among the categories with top-k highest probabilities. The top-k accuracy can thus be defined as the portion of correctly classified images among all test images, and we use $k = 5$ for the energy-efficient DNN training benchmark. Since DNNs for image classification are usually a CNN model with limited types of operations and relatively simple network structures, training the image classification models on FPGAs should be the easiest task to achieve.

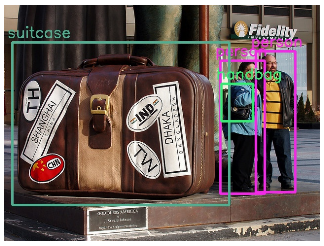

Figure 2. The object detection result of an example image from the COCO dataset [28]

Object detection is a more complex computer vision task than image classification. As is shown in Figure 2, the objective of object detection is to find the locations of the semantic objects in images using bounding boxes. Two of the most widely used object detection datasets are the Pascal Visual Object Classes (VOC) dataset (with 27,450 annotated objects from 11,530 images) [10] and the Common Objects in COntext (COCO) dataset [28] which is a larger dataset with 1.5 million annotated objects from 330K images. The Intersection over Union (IoU) is often used to

Table 2. Summary of the identified computer visions tasks for the energy-efficient DNN training benchmark

Tasks	**Pre-Training Dataset**	**Fine-Tuning and Test Dataset**	**Task-Specific Performance Metric**	**Example Solution**
Image Classification	ImageNet	Food101	Top-5 Accuracy	ResNet [18]
Object Detection	COCO	Pascal VOC	mAP	Faster R-CNN [36]
Video Classification	Moments in Time	UCF-101	Top-5 Accuracy	Slow Fusion [23]

measure how well an algorithm performs on detecting an object (as defined below).

$$IoU = \frac{A \cap B}{A \cup B} \tag{1}$$

where A is the predicted bounding box of an object in the image and B is the ground truth bounding box. $A \cap B$ represents the intersected area between these two bounding boxes, whereas $A \cup B$ is their combined area. The IoU is then obtained by dividing these two areas.

Mean average precision (mAP) is the most commonly used metric to evaluate the overall performance of an object detection algorithm and is defined as follow.

$$mAP = \frac{1}{N} \sum_{i}^{N} AP_i \tag{2}$$

where mAP is the average AP among all classes and AP_i stands for the average precision of all objects in the i_{th} class. The value of AP can be derived by calculating the area under the precision-recall curve (described in [10]). An IoU threshold is used to determine whether an object is correctly detected or not. For example, $AP^{0.5}$ means the detected objects with IoU scores of above 0.5 would be considered as correct. The DNN architectures for object detection may contain more types of operations and generate the outputs of different sizes, which need to be handled accordingly in the hardware design. The COCO dataset will be used to pre-train the object detection model, and the data in the Pascal VOC dataset are used for fine-tuning and testing.

Similar to image classification, video classification aims at assigning a class label to a video clip with a stack of images. In this benchmark, we mainly focus on a subcategory of video classification, namely action recognition, where the class labels are related to actions. The two most commonly used datasets of action recognition are UCF-101 [40] dataset (with 13,320 videos from 101 categories) and Moments-In-Time [30] which a large dataset with 1 million videos. For video classification, the DNN might include RNN or GNN structures that have not been implemented on FPGAs with CNNs. Such tasks bring in more challenges to manage the on-chip resources and hardware design, which is beyond the limit of current techniques. In this benchmark, Moments-In-Time dataset will be used for pre-train the model while the data in UCF-101 that share the same categories as the Moments-In-Time dataset will be used for fine-tuning and testing.

4.2. Evaluation Workflow and Performance Metric

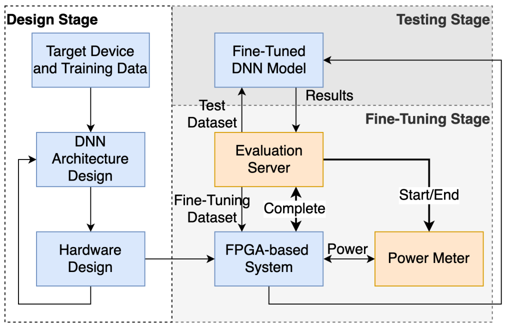

Figure 3. The proposed evaluation workflow

In addition to task-specific performance metrics, the system performance should be measured, including power consumption, processing time, and resource usage. Figure 3 depicts the diagram of how the system evaluation is conducted, which includes three stages, namely the design stage, fine-tuning stage, and testing stage. In the design stage, for a specific task, the DNN architecture and its hardware design for a target device should be conducted. The design might go through several cycles, i.e., the DNN architecture needs to be updated based on the results of the hardware design. Once the design stage is completed, a DNN architecture, the corresponding FPGA-based hardware design for model training, and the pre-trained weights of the DNN model can be obtained. After that, the DNN training framework will be deployed on the FPGA-based system, where the fine-tuned dataset will be fed to the embedded system to fine-tune the model. Once the model is being trained, the power meter connected to the system will start to measure the real-time power consumption of the system. A complete signal will be sent to the tested system when a maximum training time T_{max} is reached. Alternatively, if the fine-tuning process is completed within the time limit, the system will send a complete signal to the evaluation server upon completion. In this case, the total training time will be recorded. On the other hand, once the training is completed, the evaluation server will send a stop signal to the power meter to terminate the power measurement. The amount of power consumed during the fine-tuning process will be accumulated and used as the metric for energy-efficiency. After the fine-tuning step, the evaluation server will send the test dataset to evaluate the fine-tuned model. The system will be switched into the inference mode and then make predictions on this dataset. Finally, the predicted results will be sent back to the evaluation server for task-specific performance evaluation. The evaluation system, including both fine-tuning and testing stages, can be deployed on Amazon EC2 F1 instances[2]. Deploying the evaluation system on the cloud can allow for flexible accesses to the system and improve the scalablility.

To evaluate the overall quality of the design, a comprehensive metric should be adopted. Such a performance metric should consider all aspects of the design, including on-chip resource usage, training efficiency, energy efficiency, and task-specific model performance. Among various resources available on FPGAs, the most important ones for DNN training include the look-up table (LUT), digital signal processor (DSP), block RAM (BRAM), and flip-flop (FF). The average utilization rate of these resources (denoted by M_u) can be used for evaluating the on-chip resource usage. If the same FPGA is used for evaluation, the lower the utilization rate, the better the design quality.

Since the total training time is recorded during the evaluation, the training efficiency can be represented by

$$M_t = \frac{T}{T_{max}} \tag{3}$$

where T is the time taken to train the model.

Regarding energy efficiency, it is common in the hardware design to specify an energy budget P_{max}. For various application scenarios, P_{max} can have different values, and 10W can be a suitable candidate for the mobile computing. Given the energy budget, if a solution exceeds the budget, it fails automatically. Otherwise, the score for energy efficiency can be computed by

$$M_p = \frac{W_{total}}{P_{max} \times T} \tag{4}$$

where W_{total} is the total power consumption during the fine-tuning stage, which is measured by the power meter.

The task-specific metric M_s can be the top-k accuracy value for the image and video classification tasks and the mAP value for the object detection task. Hence, the overall system performance can be scored as

$$S = 1 - [w_u M_u + w_t M_t + w_p M_p + w_s(1 - M_s)] \tag{5}$$

where w_u, w_t, w_p, and w_s are the weighting factors of each individual metric respectively and $w_u + w_t + w_p + w_s = 1$. The overall system performance S ranges from 0 to 1, and the design quality is better when S is closer to 1. Since the resource usage will be implicitly reflected in energy efficiency M_p and training efficiency can be partially reflected in model performance M_s, it is desired to assign lower values to w_u and w_t.

[2]`https://aws.amazon.com/ec2/instance-types/f1/`

5. Conclusion

In this paper, the advantages and disadvantages of deploying DNNs on FPGAs compared to CPUs and GPUs are summarized. FPGA is an alternative platform to train DNNs at the edge with the constraints of energy efficiency. However, the solutions to train DNNs on the edge devices using FPGAs have not been well exploited. The challenges of implementing DNN training on FPGAs mainly lie in the complexity of resource management and the requirements of both software and hardware design knowledge to obtain a valid solution. Therefore, we propose to leverage the automated hardware design based on HLS to accelerate the development cycle and to achieve a better system performance. To evaluate the performance of the FPGA-based DNN training systems and motivate the research in this domain, a comprehensive performance metric is proposed with the consideration of on-chip resource usage, training efficiency, energy efficiency, and model accuracy. We have identified three critical computer vision tasks, namely image classification, object detection, and video classification, and design an evaluation workflow to measure the design quality of the solutions for these three tasks.

References

[1] Sergei Alyamkin, Matthew Ardi, Alexander C Berg, Achille Brighton, Bo Chen, Yiran Chen, Hsin-Pai Cheng, Zichen Fan, Chen Feng, Bo Fu, et al. Low-power computer vision: Status, challenges, and opportunities. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(2):411–421, 2019.

[2] Sajid Anwar and Wonyong Sung. Compact deep convolutional neural networks with coarse pruning. *CoRR*, abs/1610.09639, 2016.

[3] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101 - mining discriminative components with random forests. In David J. Fleet, Tomás Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *European Conference on Computer Vision*, volume 8694 of *Lecture Notes in Computer Science*, pages 446–461, 2014.

[4] Han Cai, Chuang Gan, and Song Han. Once for all: Train one network and specialize it for efficient deployment. *CoRR*, abs/1908.09791, 2020.

[5] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019.

[6] Jason Cong, Zhenman Fang, Michael Lo, Hanrui Wang, Jingxian Xu, and Shaochong Zhang. Understanding performance differences of FPGAs and GPUs. In *IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, pages 93–96, 2018.

[7] Jason Cong, Bin Liu, Stephen Neuendorffer, Juanjo Noguera, Kees A. Vissers, and Zhiru Zhang. High-level synthesis for fpgas: From prototyping to deployment. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 30(4):473–491, 2011.

[8] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Training deep neural networks with low precision multiplications. *CoRR*, abs/1412.7024, 2014.

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[10] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John M. Winn, and Andrew Zisserman. The pascal visual object classes (VOC) challenge. *Int. J. Comput. Vis.*, 88(2):303–338, 2010.

[11] Julian Faraone, Martin Kumm, Martin Hardieck, Peter Zipf, Xueyuan Liu, David Boland, and Philip H. W. Leong. Addnet: Deep neural networks using fpga-optimized multipliers. *IEEE Trans. Very Large Scale Integr. Syst.*, 28(1):115–128, 2020.

[12] Tong Geng, Tianqi Wang, Ahmed Sanaullah, Chen Yang, Rui Xu, Rushi Patel, and Martin C. Herbordt. Fpdeep: Acceleration and load balancing of CNN training on FPGA clusters. In *IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, pages 81–84, 2018.

[13] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

[14] Kaiyuan Guo, Shulin Zeng, Jincheng Yu, Yu Wang, and Huazhong Yang. A survey of FPGA based neural network accelerator. *CoRR*, abs/1712.08934, 2017.

[15] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In Yoshua Bengio and Yann LeCun, editors, *International Conference on Learning Representations*, 2016.

[16] Cong Hao, Xiaofan Zhang, Yuhong Li, Sitao Huang, Jinjun Xiong, Kyle Rupnow, Wen-Mei Hwu, and Deming Chen. FPGA/DNN co-design: An efficient design methodology for iot intelligence on the edge. In *Proceedings of the 56th Annual Design Automation Conference*, page 206, 2019.

[17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *IEEE International Conference on Computer Vision*, pages 2980–2988, 2017.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[19] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. AMC: automl for model compression and acceleration on mobile devices. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *European Conference on Computer Vision*, volume 11211, pages 815–832, 2018.

[20] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems*, pages 4107–4115, 2016.

[21] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and $<$1mb model size. *CoRR*, abs/1602.07360, 2016.

[22] Weiwen Jiang, Xinyi Zhang, Edwin Hsing-Mean Sha, Lei Yang, Qingfeng Zhuge, Yiyu Shi, and Jingtong Hu. Accuracy vs. efficiency: Achieving both through FPGA-implementation aware neural architecture search. In *Annual Design Automation Conference*, page 5, 2019.

[23] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[24] Jakub Konecný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *CoRR*, abs/1610.05492, 2016.

[25] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images, 2009.

[26] Griffin Lacey, Graham W. Taylor, and Shawki Areibi. Deep learning on FPGAs: Past, present, and future. *CoRR*, abs/1602.04283, 2016.

[27] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[28] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision*, pages 740–755, 2014.

[29] Markets and Markets. Computer vision market by component (hardware and software, product, application, vertical) - global forecasts to 2023. `https://www.marketsandmarkets.com/Market-Reports/computer-vision-market-186494767.html`. Accessed: 2020.04.16.

[30] Mathew Monfort, Carl Vondrick, Aude Oliva, Alex Andonian, Bolei Zhou, Kandan Ramakrishnan, Sarah Adel Bargal, Tom Yan, Lisa M. Brown, Quanfu Fan, and Dan Gutfreund. Moments in time dataset: One million videos for event understanding. *IEEE Trans. Pattern Anal. Mach. Intell.*, 42(2):502–508, 2020.

[31] Eriko Nurvitadhi, Ganesh Venkatesh, Jaewoong Sim, Debbie Marr, Randy Huang, Jason Ong Gee Hock, Yeong Tat Liew, Krishnan Srivatsan, Duncan Moss, Suchit Subhaschandra, et al. Can fpgas beat gpus in accelerating next-generation deep neural networks? In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 5–14, 2017.

[32] Samira Pouyanfar, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria E. Presa Reyes, Mei-Ling Shyu, Shu-Ching Chen, and S. S. Iyengar. A survey on deep learn-

ing: Algorithms, techniques, and applications. *ACM Comput. Surv.*, 51(5):92:1–92:36, 2019.

[33] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, et al. Going deeper with embedded fpga platform for convolutional neural network. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 26–35, 2016.

[34] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In *AAAI Conference on Artificial Intelligence*, pages 4780–4789, 2019.

[35] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In *AAAI Conference on Artificial Intelligence*, pages 4780–4789. AAAI Press, 2019.

[36] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, pages 91–99, 2015.

[37] Christopher De Sa, Megan Leszczynski, Jian Zhang, Alana Marzoev, Christopher R. Aberger, Kunle Olukotun, and Christopher Ré. High-accuracy low-precision training. *CoRR*, abs/1803.03383, 2018.

[38] Hardik Sharma, Jongse Park, Divya Mahajan, Emmanuel Amaro, Joon Kyung Kim, Chenkai Shao, Asit Mishra, and Hadi Esmaeilzadeh. From high-level deep neural models to fpgas. In *Annual IEEE/ACM International Symposium on Microarchitecture*, pages 17:1–17:12, 2016.

[39] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[40] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. *CoRR*, abs/1212.0402, 2012.

[41] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. In Anna Korhonen, David R. Traum, and Lluís Màrquez, editors, *Conference of the Association for Computational Linguistics*, pages 3645–3650, 2019.

[42] Naveen Suda, Vikas Chandra, Ganesh Dasika, Abinash Mohanty, Yufei Ma, Sarma Vrudhula, Jae-sun Seo, and Yu Cao. Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 16–25, 2016.

[43] Haiman Tian, Shu-Ching Chen, and Mei-Ling Shyu. Genetic algorithm based deep learning model selection for visual data classification. In *IEEE International Conference on Information Reuse and Integration for Data Science*, pages 127–134. IEEE, 2019.

[44] Yuexuan Tu, Saad Sadiq, Yudong Tao, Mei-Ling Shyu, and Shu-Ching Chen. A power efficient neural network implementation on heterogeneous FPGA and GPU devices. In *IEEE International Conference on Information Reuse and Integration for Data Science*, pages 193–199. IEEE, 2019.

[45] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[46] Shreyas Kolala Venkataramanaiah, Yufei Ma, Shihui Yin, Eriko Nurvitadhi, Aravind Dasu, Yu Cao, and Jae-sun Seo. Automatic compiler based FPGA accelerator for CNN training. In Ioannis Sourdis, Christos-Savvas Bouganis, Carlos Álvarez, Leonel Antonio Toledo Díaz, Pedro Valero-Lara, and Xavier Martorell, editors, *International Conference on Field Programmable Logic and Applications*, pages 166–172, 2019.

[47] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019.

[48] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *CoRR*, abs/1901.00596, 2019.

[49] Qingcheng Xiao, Yun Liang, Liqiang Lu, Shengen Yan, and Yu-Wing Tai. Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on fpgas. In *Proceedings of the 54th Annual Design Automation Conference 2017*, pages 1–6, 2017.

[50] Guandao Yang, Tianyi Zhang, Polina Kirichenko, Junwen Bai, Andrew Gordon Wilson, and Christopher De Sa. SWALP : Stochastic weight averaging in low precision training. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *International Conference on Machine Learning*, volume 97, pages 7015–7024, 2019.

[51] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 161–170, 2015.

[52] Chen Zhang, Guangyu Sun, Zhenman Fang, Peipei Zhou, Peichen Pan, and Jason Cong. Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 38(11):2072–2085, 2019.

[53] Min Zhang, Linpeng Li, Hai Wang, Yan Liu, Hongbo Qin, and Wei Zhao. Optimized compression for implementing convolutional neural networks on fpga. *Electronics*, 8(3):295, 2019.

[54] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856. IEEE Computer Society, 2018.

[55] Ritchie Zhao, Weinan Song, Wentao Zhang, Tianwei Xing, Jeng-Hau Lin, Mani Srivastava, Rajesh Gupta, and Zhiru Zhang. Accelerating binarized convolutional neural networks with software-programmable fpgas. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 15–24, 2017.

[56] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.