Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 141 (2018) 104–111

Procedia Computer Science

www.elsevier.com/locate/procedia

The 9th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2018)

A Hybrid GPU-FPGA-based Computing Platform for Machine Learning

Xu Liu[a,*], Hibat Allah Ounifi[a], Abdelouahed Gherbi[a], Yves Lemieux[b], Wubin Li[b]

[a] *Department of Software and IT Engineering*
École de Technologie Supérieure (ÉTS), Montréal, Canada
{xu.liu.1, hibat-allah.ounifi.1}@ens.etsmtl.ca, abdelouahed.gherbi@etsmtl.ca
[b] *Ericsson Research, Ericsson, Montréal, Canada*
{yves.lemieux, wubin.li}@ericsson.com

Abstract

We present a hybrid GPU-FPGA based computing platform to tackle the high-density computing problem of machine learning. In our platform, the training part of a machine learning application is implemented on GPU and the inferencing part is implemented on FPGA. It should also include a model transplantation part which can transplant the model from the training part to the inferencing part. For evaluating this design methodology, we selected the LeNet-5 as our benchmark algorithm. During the training phase, GPU TitanXp's speed was about 8.8x faster than CPU E-1620 and in the inferencing phase, FPGA Arria-10's inferencing speed was fastest, 44.4x faster than CPU E-1620 and 6341x faster than GPU TitanXp. Moreover, by adopting our design methodology, we improved our LeNet-5 machine learning model's accuracy from 99.05% to 99.13%, and successfully preserved the accuracy (99.13%) when transplanting the model from the GPU platform to the FPGA platform.

Keywords: GPU; FPGA; hybrid computing; convolution neural network; machine learning; model transformation; heterogeneous platform;

1. Introduction

The core computing jobs of machine learning are matrix operations which can be turned into many simple computing works. General purpose processors such as CPUs which have complex instruction system and execute instructions in the sequence are good at processing small amount complex logical controlling jobs but are weak for high-density computing jobs. Moreover, people find increasing training data will improve the model's accurate, more and more training data overwhelm the general purpose processors.

* Xu Liu. Tel.: +0-000-000-0000 ; fax: +0-000-000-0000.
E-mail address: xu.liu.1@ens.etsmtl.ca

10.1016/j.procs.2018.10.155

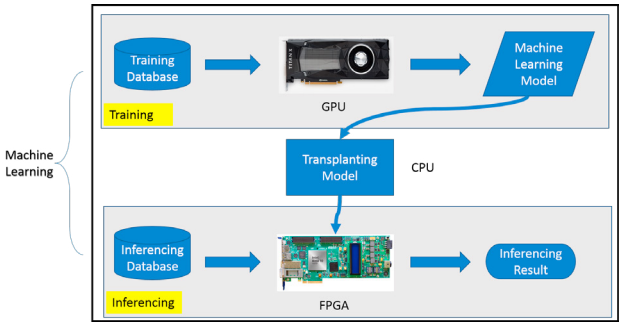

Fig. 1. The architecture of the hybrid machine learning system.

For reversing the shortage of computing power, people turn to some special hardware devices such as GPUs and FPGAs for help. Both GPUs and FPGAs have a lot of computing units which can be programmed to work in parallel to increase the speed of machine learning.

A whole machine learning should include two main phases, a training phase, and an inferencing phase. Sometimes there also needs a model file transformation phase.

Currently, most works revolve around how to use the GPUs to accelerate the training phase[14, 4, 15, 12] or how to use the FPGAs to accelerate the inferencing phase[10, 13, 11, 2], however, few persons think about how to combine the GPUs and FPGAs together to improve the performance of machine learning, and investigate what kind of hardware devices combination has the best performance, let alone how to transplant a machine learning model from one platform to another.

After investigating several kinds of GPU and FPGA combination, we found the combination of using the GPU to train model and using the FPGA to do inferencing has a better performance. To evaluate this methodology's actual performance, we created a hybrid machine learning platform whose training phase implemented on GPU and inferencing phase implemented on FPGA. This platform is easy to program and can dramatically improve the performance of training and inferencing.

2. Related Work

To the best of our knowledge, there are only three papers which combine FPGA and GPU together to improve the performance of machine learning systems. The first work we found is an FPGA-GPU architecture for kernel SVM pedestrian detection by Sebastian et al. [3]. They used GPU for model training and inferencing and used FPGA for features extraction. Maohua et al. [20] implemented a novel parallel framework for neural networks using GPU and FPGA. In their work, the neural network processing is decomposed into layers and scheduled either on the GPU or FPGA accelerators. Additionally, in a white paper [1] by Advanced Micro Devices, Inc. and Xilinx, Inc., without giving detailed information, the authors predicted that the combination of CPU, GPU and FPGA will have the best performance. And the model transplantation tech among different platforms has not been mentioned in any of their works.

3. Design Methodologies of the Hybrid Machine Learning Platform

3.1. The Architecture and Workflow

Fig. 1 presents the architecture of our hybrid machine learning system which contains three parts, i.e., the training part, the inferencing part, and model transplantation part. The training part is implemented on the GPU and in charge of training the model from the database. The inferencing part is built by FPGA and responsible for inferencing. As the FPGA inferencing part cannot load the model generated by GPU training part directly, a transplantation component is added.

3.2. The GPU Training Part

The goal of the training phase is to train a model to obtain the maximum accuracy within minimum time. Based on two reasons, we select the GPU to implement the training part. One reason is the training part focuses on fast and accurate building model. And most training jobs are done only one time. So, during this phase, high-density computing ability is more important than computing latency. At the same price level, GPUs have more computing cores and are more efficient in model training[16, 14, 12], whereas FPGAs whose resources are limited and expensive are rarely used for model training[19]. The other reason is the GPUs are easier to program than FPGAs. As the requirements of the training part are usually changed, easy programming is more important. Based on the CUDA(Compute Unified Device Architecture) many companies have already developed a series frameworks such as TensorFlow which is an open-source software framework for machine learning developed by Google Brain[6] have already supported GPU acceleration. These frameworks integrate CUDA functions and hide CUDA implementation details, allowing users to focus on the design of machine learning algorithms.

3.3. FPGA Inferencing Part

The goal of the inferencing phase is to do inferencing with the model generated by the training phase with minimum latency. The inferencing phase will often be done time and time again within a period. Any small latency of each cycle will accumulate to a huge amount. So it is meaningful to reduce the inferencing latency. FPGAs are composed of logical elements(LEs) which can be programmed into hardware logic circuits. And in most FPGA implementations, there is no instruction fetching and decoding system, this will make FPGAs run much faster than GPUs or CPUs.

However, FPGAs have some disadvantages such as limit and expensive resources and hard development. Limited by the complex and difficult manufacturing process, the price of same area size FPGAs is usually more expensive than GPUs. This limits the FPGAs' area size could not be too big. And in the early stage, the FPGAs can only be developed by HDL(hardware description languages) such as Verilog or VHDL(VHSIC (Very High-Speed Integrated Circuit) Hardware Design Language) which require having deep knowledge of logical electronic circuits and system. Similar to the assembly languages, even a small function needs a lot of HDL sentences to describe. If the system is too complex like deep neuro-network, it is hard to use HDLs to develop it. Now, the difficulties of FPGA design can be compensated by using high-level programming languages such as OpenCL(Open Computing Language) which do not require much hardware knowledge [2, 19, 5, 9]. OpenCL is a framework. It can be used to develop programs which can be run on different heterogeneous platforms such as GPUs(graphics processing units) and FPGAs(field-programmable gate arrays).

3.4. Model Transplantation

For efficiency, most machine learning models will not be run the platforms which generate them. So a model transplantation phase is needed. Although there are already some tools such as MMdnn for transforming the model file among the main frameworks, it is not fit for our work. As our inferencing phase is implemented by FPGA, all functions are built from the scratches, its framework is different from any other main frameworks.

A machine learning model file records parameters such as weights and biases generated during the model training phase and will be loaded during the inferencing phase. Fig 2 is a simple model file transformation example. It contains a convolution and a flatten operations. For this example, the decimal points and flatten ways are different. W_1 in model file 1 is flattened as $[1, 1, 0, 0]$ while W_2 in model file 2 is flattened as $[0.1, 0, 0.1, 0]$. So the model 1 of framework 1 cannot be used directly on framework 2, it should do model transformation firstly.

For a real machine learning model transformation, the situation would be more complex, the parameters even have four dimensions. There are 256 ways to flatten a four dimension parameter and only one way works for the specific model. So it is a difficult and non-obvious job for model transformation.

4. A Use Case of the Hybrid Machine Learning Platform

To verify our platform design practices, we implemented a LeNet-5 machine learning algorithm in accordance with our platform design methodologies.

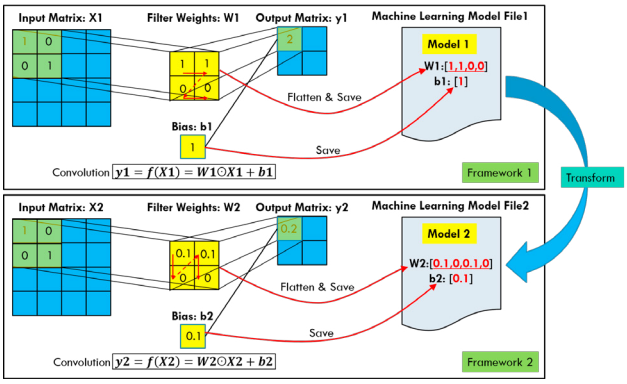

Fig. 2. Schematic of Model Transformation.

4.1. The overview of the Use Case

4.1.1. The Hardware Environment

Our devices' list is shown in Table 1. The Titan XP graphics card communicates with the host machine through PCIe gen 3 x16 (with the bandwidth of 15760 MB/s). The Arria 10 development board transfers data with the host machine by PCIe gen 3 x8 (with the bandwidth of 7880 MB/s). The amount of training data which is usually huge and is divided into many small batches cannot be loaded into GPU in one-time. So the data will be frequently transferred between the host and the device. On the contrary, each inference which contains small data no need to transfer data frequently.

From this viewpoint, the bandwidth becomes a performance bottleneck of the system, so it is a better choice to select the GPU to do the training and the FPGA to do the inferencing.

Table 1. Hardware Devices List.

Device	Type	Number
CPU	Intel(R) Xeon(R) CPU E5-1620 v4 @ 3.50GHz	1
Memory	Samsung DDR4 8g	4
Solid State Disk Drive	INTEL SSDSC2BB48 480g	1
Mechanical Hard Disk Drive	WDC WD40EFRX-68N 4TB	1
GPU	NVIDIA TITAN Xp	1
FPGA	Intel Arria 10 GX FPGA Development Kit	1

4.1.2. The Software Environment

Our software environment is as shown in Table 2. The operating system is Ubuntu 16.04.

Table 2. Software Environment.

Software	Version
Ubuntu	16.04.3 LTS
Python	3.5.2
Tensorflow	1.4.0
CUDA	8.0
Intel(R) FPGA SDK for OpenCL	17.1.0 Build 240

4.2. The Algorithm of the Use case

As our goal is to verify the performance of the hybrid machine learning system, we select LeNet-5 and MNIST as our algorithm and dataset, which are not too complex but enough to show the effects of hardware acceleration.

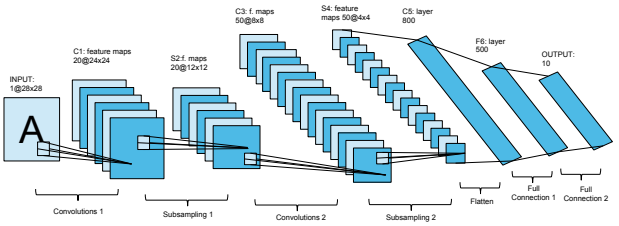

Fig. 3. The Architecture of LeNet-5 Improved for Experiment.

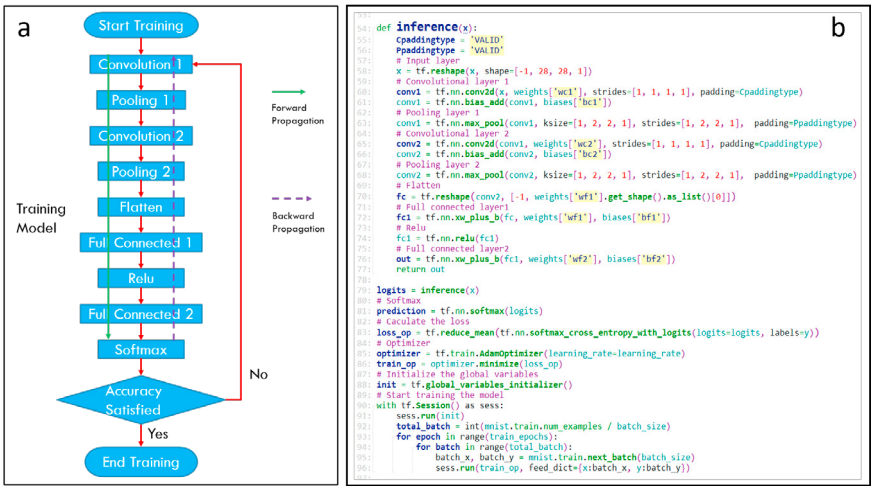

Fig. 4. (a) The Control Flow of Training; (b) The TensorFlow Implementation of the Training Part

The MNIST[8] is a training set for digital handwriting recognition. Each example is a pixel value matrix whose size is 28 x 28 and each pixel value's range is from 0 to 255. In some cases, for easy processing, people divide the pixel value by 255. In total, MNIST includes 55,000 training examples, 5,000 validation examples, and 10,000 test examples.

The LeNet-5 is a convolution neural network designed by Yann LeCun for handwritten and machine-printed character recognition [7]. It has almost all the structures(convolution layers, pooling layers, and full connection layers, etc.) of classical convolution neural network and has a high accuracy in digital handwriting recognition and most importantly, it contains a lot of high-density computing jobs which are suitable for testing the acceleration effects of parallel design. So it is an idea machine learning algorithm for evaluating hardware acceleration.

Fig. 3 is our version of LeNet-5 used for this implementation. It has 7 layers totally, 3 convolution layers, 2 sub-sampling layers and 2 full connection layers. These layers are orderly arranged as illustrated in the Fig. 3.

4.3. The Training Part of the Use Case

4.3.1. The Implementation

Fig. 4(a) is our training control flow. It includes two main phases, i.e. forward propagation and backward propagation. The difference of value calculated by the forward propagation and label value is passed into the backward propagation to calculate the weights and biases of each layer. After several computing loops, the difference converges within a small range, the training process terminates. The final weights and bias will be stored in a model file.

We used NVIDIA Titan Xp graphics card to accelerate the training process. The GPU has 3840 CUDA cores which can be programmed to do parallel computing. Its floating computing performance can reach 12 TFLOPS.

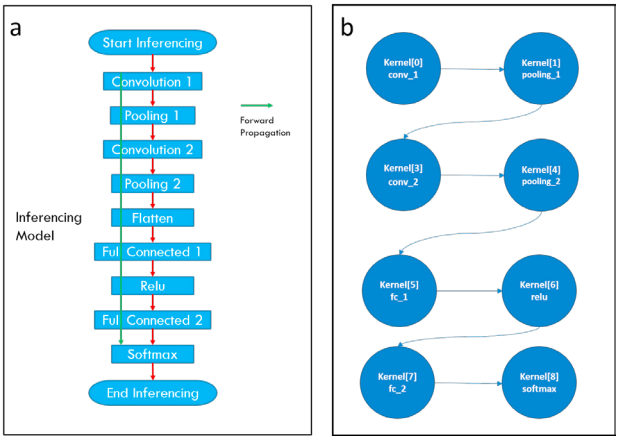

Fig. 5. (a) The Control Flow of Inferencing; (b) The Kernels of OpenCL Implementation

To do the NVIDIA graphics card computing development, it should use the CUDA programming language mentioned previously. That will mean you should build the acceleration kernel from scratch. Thanks to the TensorFlow, which packages the CUDA libraries, now we only need to focus on designing the architecture. The Fig. 4(b) is part of our TensorFlow implementation code.

4.3.2. The Experiment Results

In this experiment, our aim is to find the best device for training model by comparing the speed of CPU and GPU.

For achieving this goal, we designed two use cases. One only has a CPU-E5-1620, the other has a CPU-E5-1620 and a GPU-TitanXp. We chose the same 55000 training examples and calculated their training time respectively. We did six times the same tests and calculated their average values.

Table 3. The Training Time.

Experiment Times	CPU-E5-1620		GPU-TitanXp		Accelerate Times
	Training Time(*s*)	Accuracy (*%*)	Training Time (*s*)	Accuracy (*%*)	Acceleration(*GPU/CPU*)
1	448.60	98.70	50.61	98.80	8.86
2	447.79	98.88	51.17	98.58	8.75
3	448.35	98.90	50.73	98.80	8.84
4	448.62	98.94	50.46	98.88	8.89
5	447.62	98.59	50.94	98.82	8.79
6	447.88	98.94	50.78	98.79	8.82
Average	448.10	98.80	50.80	98.80	8.80

The results are in Table 3. We can get a conclusion that the average speed of TitanXp is about 8.8x faster than the average speed of CPU E5-1620 with the similar accuracy. This result is similar to Tobias et al.' work [17] whose GPU speed is about 9x faster than CPU.

4.4. The Inferencing Part of the Use Case

4.4.1. The Implementation

Fig. 5(a) is our inferencing phases control flow. It only has forward propagation. It loads the model generated from the training part and then does inferencing with the forward propagation algorithm.

We used Intel Arria 10 FPGA development board to implement the inferencing part. Arria 10 is the newest FPGA product made by Intel with 20nm technique. It has high performance and low power consumption. Although its 1.5

TFLOPS is still slower than TITAN Xp 12 TFLOPS, its power consumption below 100 watts is much better than TITAN Xp 250 watts.

We used OpenCL to do the FPGA development. Its development includes two parts: host program and kernels. The host program runs on the CPU and the kernels run on the FPGA. Fig. 5(b) shows our kernels implemented with OpenCL. The kernels are mapped with hardware logical circuits individually and executed one by one to implement the inferencing function.

4.4.2. The Experiment Results

In this experiment, we used the CPU, GPU, and FPGA to do the same inferencing work and compare their efficiencies.

For avoiding the impacts from different bandwidths, we only calculated the time of inferencing one image. We executed 6 times inferencing on the CPU, GPU, and FPGA devices and calculated the average time values respectively.

The results of the experiments are presented in Table 4.

Table 4. The Inferencing Time.

Experiment Times	CPU-E5-1620	GPU-TitanXp		FPGA-Arria10		
	Inferencing Time (us)	Inferencing Time (us)	Acceleration (GPU/CPU)	Inferencing Time (us)	Acceleration (FPGA/CPU)	Acceleration (FPGA/GPU)
1	3172	616045	0.0051	88.74	35.7	6942.4
2	5564	589114	0.0094	90.12	61.7	6536.7
3	4620	588444	0.0079	94.83	48.7	6205.3
4	3234	598652	0.0054	84.57	38.2	7079.0
5	4037	600288	0.0067	101.08	39.9	5938.8
6	4579	609913	0.0075	108.74	42.1	5609.0
Average	4201	600409	0.0070	94.70	44.4	6341.5

From Table 4, we can see that the average speed of Arria 10 is about 44.4 times faster than the average speed of CPU E5-1620 and is about 6342 times faster than the GPU TitanXp. During Wang et al.'s work [18], FPGA is 36.1x faster than CPU. These results prove our decision to use the FPGA to do inferencing is right. And for the GPU's worst behavior, our explanation is that as the workload(inferencing one image) is too small compared with the GPU initialization time and delay, the total time(initialization time and inferencing time) of GPU is longest.

4.5. The Model transplantation of the Use Case

4.5.1. The Experiment Results

We conducted two experiments on the same 10K MNIST test examples respectively. In Experiment_1, we only test FPGA inferencing accuracy with the original model while in Experiment_2, we changed CNN's configuration, retrained the model by TensorFlow with GPU, transplanted the model to the FPGA and then tested the inferencing accuracy on FPGA and Tensorflow.

Table 5 presents the statistics of accuracies collected from the two experiments. The Experiment_2's accuracy is better than Experiment_1's. In Experiment_2, the FPGA has preserved the same accuracy as the TensorFlow, which proved our model transplantation successfully.

Table 5. The Accuracies in Model Transformation Experiments.

Experiment_1	Accuracy(%)	Experiment_2	Accuracy(%)
N/A	N/A	TensorFlow	99.13
FPGA	99.05	FPGA	99.13

5. Conclusion and Future Work

We suggested a hybrid machine learning platform which included three main components: training with GPU, inferencing with FPGA, and model transplantation with CPU. We evaluated the design by implementing a LeNet-5 machine learning algorithm on our platform. The experiment results show that the GPU training speed is average 8.8x faster than the CPU and the FPGA inferencing speed is average 44.4x faster than the CPU and is average 6342x faster than the GPU. And we improved the model's accuracy from 99.05% to 99.13% and preserved the accuracy 99.13% successfully when we transplanted the model from GPU platform to the FPGA platform.

Regarding the future work, we will do more investigation on the energy consumption of our platform.

Acknowledgements

This work is partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and Ericsson Research.

References

[1] Advanced Micro Devices, Inc. and Xilinx, Inc., 2017. Unified Deep Learning with CPU, GPU, and FPGA Technologies. White Paper.
[2] Aydonat, U., O'Connell, S., Capalija, D., Ling, A.C., Chiu, G.R., 2017. An OpenCL Deep Learning Accelerator on Arria 10., in: Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, ACM. pp. 55–64.
[3] Bauer, S., Köhler, S., Doll, K., Brunsmann, U., 2010. FPGA-GPU Architecture for Kernel SVM Pedestrian Detection, in: Proceedings of the 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), IEEE. pp. 61–68.
[4] Bergstra, J., Bastien, F., Breuleux, O., Lamblin, P., Pascanu, R., Delalleau, O., Desjardins, G., Warde-Farley, D., Goodfellow, I., Bergeron, A., et al., 2011. Theano: Deep learning on gpus with python, in: NIPS 2011, BigLearning Workshop, Granada, Spain, Citeseer.
[5] Bettoni, M., Urgese, G., Kobayashi, Y., Macii, E., Acquaviva, A., 2017. A Convolutional Neural Network Fully Implemented on FPGA for Embedded Platforms, in: New Generation of CAS (NGCAS), IEEE. pp. 49–52.
[6] Google Inc., . TensorFlow: An Open Source Machine Learning Framework for Everyone. https://www.tensorflow.org/, visited January 2018.
[7] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., 1998. Gradient-based Learning Applied to Document Recognition. Proceedings of the IEEE 86, 2278–2324.
[8] LeCun, Yann and Cortes, Corinna and J.C. Burges, Christopher, 2018. The MNIST Database. URL: http://yann.lecun.com/exdb/mnist/.
[9] Li, Y., Liu, Z., Xu, K., Yu, H., Ren, F., 2017. A GPU-outperforming FPGA Accelerator Architecture for Binary Convolutional Neural Networks .
[10] Motamedi, M., Gysel, P., Akella, V., Ghiasi, S., 2016. Design Space Exploration of FPGA-based Deep Convolutional Neural Networks, in: Proceedings of the 21st Asia and South Pacific Design Automation Conference (ASP-DAC), IEEE. pp. 575–580.
[11] Nagarajan, K., Holland, B., George, A.D., Slatton, K.C., Lam, H., 2011. Accelerating Machine-learning Algorithms on FPGAs using Pattern-based Decomposition. Journal of Signal Processing Systems 62, 43–63.
[12] Potluri, S., Fasih, A., Vutukuru, L.K., Al Machot, F., Kyamakya, K., 2011. CNN based High Performance Computing for Real Time Image Processing on GPU, in: 2011 Joint 3rd Int'l Workshop on Nonlinear Dynamics and Synchronization (INDS) & 16th Int'l Symposium on Theoretical Electrical Engineering (ISTET), IEEE. pp. 1–7.
[13] Qiu, J., Wang, J., Yao, S., Guo, K., Li, B., Zhou, E., Yu, J., Tang, T., Xu, N., Song, S., et al., 2016. Going deeper with embedded fpga platform for convolutional neural network, in: Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, ACM. pp. 26–35.
[14] Raina, R., Madhavan, A., Ng, A.Y., 2009. Large-scale Deep Unsupervised Learning using Graphics Processors, in: Proceedings of the 26th annual International Conference on Machine Learning, ACM. pp. 873–880.
[15] Sharp, T., 2008. Implementing decision trees and forests on a gpu, in: European conference on computer vision, Springer. pp. 595–608.
[16] Steinkraus, D., Buck, I., Y. Simard, P., 2005. Using GPUs for Machine Learning Algorithms, in: Proceedings of the 8th International Conference on Document Analysis and Recognition, IEEE. pp. 1115–1120.
[17] Tobias Kind, 2018. Tensorflow (TF) benchmarks. URL: https://github.com/tobigithub/tensorflow-deep-learning/wiki/tf-benchmarks.
[18] Wang, C., Gong, L., Yu, Q., Li, X., Xie, Y., Zhou, X., 2017. Dlau: A scalable deep learning accelerator unit on fpga. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 36, 513–517.
[19] Zhao, W., Fu, H., Luk, W., Yu, T., Wang, S., Feng, B., Ma, Y., Yang, G., 2016. F-CNN: An FPGA-based Framework for Training Convolutional Neural Networks, in: Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors, pp. 107–114.
[20] Zhu, M., Liu, L., Wang, C., Xie, Y., . CNNLab: a Novel Parallel Framework for Neural Networks using GPU and FPGA .