On-accelerator neural network training: a case study on FPGAs

***Abstract*—For the sake of efficient inference on CNN accelerators, neural network models are typically trained via the offline training frameworks such as Caffe and Tensorflow on general purposed processors (GPPs) first and then compiled to the target accelerators. However, the neural network computing on CNN accelerators and that on GPPs may differ due to the different hardware architecture or runtime environment in many scenarios. For instance, the CNN accelerator is implemented with approximate arithmetic circuits, or exposed to fault-prone environment or overclocked, In this cases, the dynamic behavior of the accelerator is difficult to be captured by simulation on GPPs and applying the conventional offline trained model to the accelerators directly may lead to considerable prediction accuracy loss.**

To address this problem, we propose to train the neural network models on the CNN accelerator and have the undetermined accelerator behavior learned together with the data in the same framework. Basically, we start from the offline trained model and then perform accelerator-specific or scenario-specific fine tuning to make the CNN models less sensitive to the accelerator's undetermined behavior. In addition, we further explore the on-accelerator fine-tuning approaches and implement a Caffe based on-accelerator training framework. We apply the on-accelerator training approach to a set of neural networks targeting at both an overclocked CNN accelerator and a faulty CNN accelerator on a CPU-FPGA computing platform. According to our experiments on ImageNet, the on-accelerator training can improve the top 5 accuracy and top 1 accuracy up to 11.65% and xx% when the CNN accelerator works at extreme clock. When the accelerator is exposed to a faulty environment, the top 5 and top 1 accuracy improves up to xxx% and 3.38% under the most sever fault injection.

I. Introduction

In recent years, deep neural networks especially the convolutional neural networks (CNN) have shown great performance in massive fields such as image classification, video surveillance, speech recognition, and robot vision. To ensure both higher energy-efficiency and lower processing latency, various CNN accelerators [1]–[6] have been developed and are increasingly deployed in the diverse applications. With the CNN accelerators, a neural network is usually trained using frameworks like Caffe, Pytorch and Tensorflow on general purposed processors (GPPs) first. After the training, the resulting neural network model is further quantized and compiled to the target CNN accelerator.

The implicit assumption of the offline training is that the neural network model executed on GPPs can produce equivalent result to that on the CNN accelerators. Nevertheless, there are many scenarios where the computing on CNN accelerators is different from that is calculated with GPPs. For instance, the CNN accelerator may be implemented with approximate arithmetic logic such as **xxx** for the purpose of higher performance or energy efficiency. Overclocking [7], [8] which allows certain timing errors to enable higher clock frequency is another occasion, and is already supported in commercial chips or FPGA designs **xxxx**. Some of the CNN accelerators may be exposed in error-prone environment where soft errors may affect the circuit behavior randomly. In these scenarios, it is almost impossible to have the offline training framework to aware the exact accelerator behavior. If the training framework ignores the accelerators dynamic behavior, the prediction accuracy of the resulting neural network model may degrade dramatically, which is expected according to the neural network training theory.

To address the above problems, we propose to train with the unstable accelerator to tolerate the accelerators un-deterministic behavior. Then we revisit the conventional training flow, define the interface to integrate the hardware accelerator into the Caffe framework and performs the necessary modification to the general CNN accelerator design. Finally, we take the overclocked and faultprone CNN accelerator as examples and demonstrate the potential of this training method. Basically, it helps to hide the low-level circuit influence to higher level applications. For the overclocking case, the application gets improved performance without much consideration of the side effects. For the accelerator exposed to soft errors, the system can proceed using the CNN model without dealing with the soft error.

The contribution of this work is summarized as the following aspects.

- We proposed to train for the unstable CNN accelerator such that the resulting model can learn the underlying un-deterministic circuit behavior together with the application data. With this method, the resulting system can tolerate the CNN accelerators un-deterministic behavior without hardware modification.
- We build an open-sourced end-to-end training framework based on Caffe to train for the unstable accelerator. In addition, we present the necessary modification of the general CNN accelerators to make use of the training framework.
- We take overclocking and fault-prone CNN accelerator as two examples and demonstrate the usefulness of the proposed system.

The paper is organized as follows. Section II analyzes the influence of the CNN accelerators un-deterministic behaviors when using the conventional training. Section III presents the proposed training of both data and the accelerator in the same framework. Meanwhile, the necessary modification of the general CNN accelerator for taking advantage of the training is detailed. Afterwards, we commit two case study using the proposed training system and demonstrate the usefulness of the system. Section IV briefs the related work. Section V draws the conclusion.

II. Motivation

As the major training systems typically adopt the off-line training method on CPUs and GPUs, un-deterministic behaviors of the accelerators will not be considered by default. To evaluate the influence, we take an overclocked CNN accelerator and a soft-error affected CNN accelerators as examples and investigate the influence of the unstable circuit on the prediction accuracy in this section.

As shown in Figure 1, we adopt PipeCNN [1] , an open sourced CNN accelerator, as the baseline accelerator and implement it on Xilinx KCU1500 boards. The accelerator runs at most 210 MHz safely for AlexNet. On a subset of ImageNet, we train AlexNet offline and then apply it to the accelerator. The resulting top-5 accuracy is 78.95%. Then the clock frequency is boosted to 250Mhz and 260MHz respectively, we apply the original model on the overclocked accelerator. The performance gets improved proportionally, but the accuracy drops 0.5% and 4.3% respectively.

Soft error has become an un-ignorable problem with the shrinking semiconductor feature size and we further analyze its influence on the CNN accelerator. Currently, we use a uniform distribution model to inject the SEU errors to the multiplication-accumulation operators (MAC) of the accelerator. It causes one-bit flip on random bits of the MAC results. When the error injection rate is set to be 0.0001% per MAC, applying the off-line trained model to the CNN accelerator leads to around 0.7% accuracy loss of the top 5 prediction. When error injection rate is set to be 0.001% per MAC, the prediction accuracy drops around 3%.

To gain insight on the precision drop, we also check the output of the AlexNets last layer. We compare the data and find that the data deviates from expected result slightly due to the unstable accelerator. But the accuracy loss of the model deployed on the unstable CNN accelerator cannot be ignored according to the experiments. And it is highly demanded to explore the training system and take the accelerators behavior into consideration during training for higher prediction accuracy.

III. Training for Unstable CNN accelerator

In contrast to the conventional off-line training on CPUs and GPUs, we propose to take these accelerators dynamic behaviors into consideration during training to tolerate the un-deterministic behavior of the unstable accelerators. The basic idea is to embed the CNN accelerator into the conventional training framework so that the accelerator is referenced during training. In this work, we choose Caffe as the baseline training framework because it is more natural to integrate the C/C++ based high level synthesis CNN accelerator. Based on Caffe, we further detail the required general interface to make use of the hardware accelerator in training, and introduce the necessary modifications to the CNN accelerator structure.

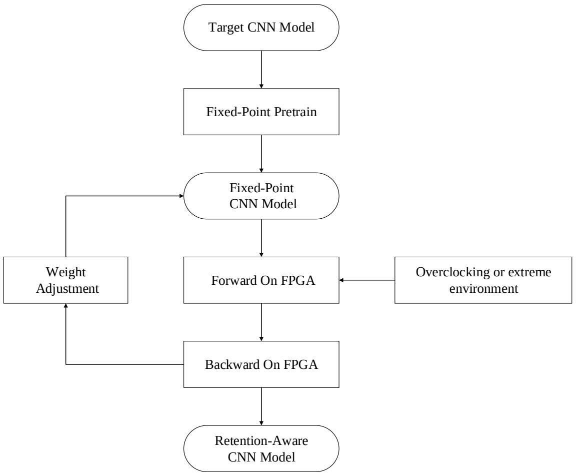

Fig. 1. Proposed Training Framework

A. Proposed Training Framework

Figure 3 illustrates the proposed training framework. It begins with the off-line training result which can greatly shorten the overall training time. While most of the CNN accelerator adopts fixed point operations, the pre-trained model is therefore expected to be fixed point model. With the pre-trained model, we mainly try to have the trained model to further adapt to the un-deterministic behaviors which are difficult to model on CPUs and GPUs.

To that end, we have the forward propagation performed on the accelerator directly while the backward propagation remains on CPUs or GPUs. Forward propagation on the accelerator is fixed point, which is beneficial to both the resource consumption and memory bandwidth overhead, but backward propagation on CPUs or GPUs remains floating point to ensure the small changes in the parameters get accumulated [9]. As a result, we still need additional converting between the fixed point and floating during the training in each iteration when the weight is adjusted. When the final accuracy loss reaches the threshold, it means that the model can tolerate the accelerators un-deterministic behaviors. And the CNN model can be safely deployed on the unstable accelerator.

Figure 4 depicts the implementation of the training framework on a hybrid CPU-FPGA architecture. In this work, we use Xilinx KCU1500 as the FPGA board and put it on a standard desktop computer. CPU is the controller and it reconfigures the accelerator for a specific CNN structure. In each training iteration, CPU launches the CNN accelerator to perform the forward propagation from bottom layer to top layer. CPU does the backward propagation from top layer to bottom layer. Weights and the image data are initially stored in host memory. It will be transferred to FPGA offchip memory for forward propagation through PCI-E. Similarly, the output data will be transferred from FPGA off-chip memory back to host memory after forward propagation. Because of the

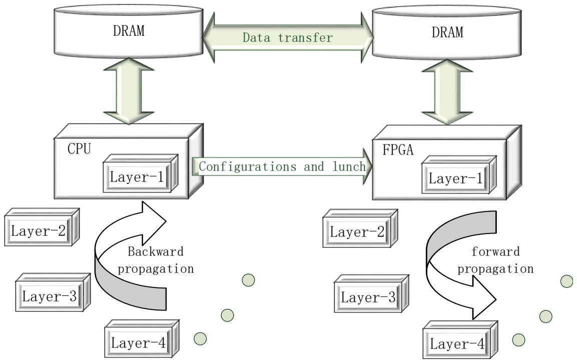

Fig. 2. Training on Hybrid CPU-FPGA Architecture

OpenCL based API wrapper in SDAccel, the CNN accelerators interface can be easily exposed to Caffe for referring to the forward propagation result.

B. High Level Accelerator Interface to Caffe

With the growing popularity of deep learning, massive different CNN accelerators have been developed over the years. In order to fit various CNN accelerators within the same training framework, we define a set of high-level interface functions as listed in Table 1. There are 7 functions included. Function 1 is used to launch the CNN accelerator from host. Function 2 and 3 are used to transfer data between the host memory and the device memory during the training. As most of the accelerators are fixed point and used for forward while back propagation is floating point, Function 4 and 5 are required for training when forward and backward propagation are iteratively committed. Function 1 to 5 are required for all the accelerators. Function 6 and 7 are only needed for accelerators that perform on reorganized data [1], [4]. With the interface functions, general CNN accelerators can be trained to tolerate un-deterministic circuit behaviors using the proposed framework. CNN accelerators can either be implemented using high-level synthesis tools (HLS) or hardware description languages (HDL). With Xilinx SDAccel, we can wrap the both types of accelerators with OpenCL API. With the OpenCL API, Caffe can refer to the accelerators during training conveniently.

C. Modification to the general CNN accelerators

On top of the interface, the CNN accelerator also needs minor adjustment for the training. The training process requires the feature maps of each CNN layer for backward propagation, while the accelerators are typically optimized for inference and some of the layers output are fully buffered in on-chip memory for less memory access overhead. In this case, the accelerator should make intermediate output write back optional as shown in Figure 5. When the accelerator is used in training, the output will be transferred to memory. When it is used for inference, it can also turn off the write back data path for better performance. It is trivial to modify the CNN accelerators and the hardware overhead is negligible.

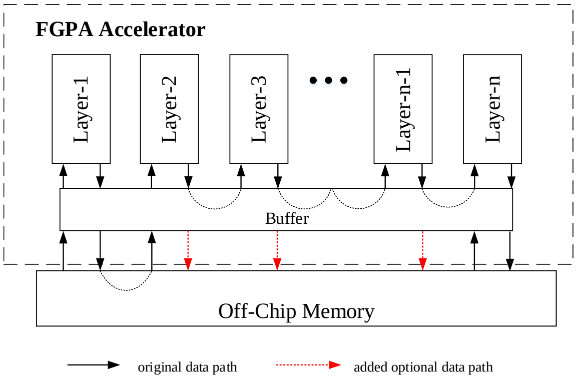

Fig. 3. Modification of the CNN accelerator data path. It essentially ensures each CNN layer to have an optional data path to off-chip memory so that it can be used for training as necessary.

IV. Case Study and Experiments

In this section, we mainly explore deploying the CNN models on the unstable accelerators using the proposed training framework. In particular, we take an overclocked CNN accelerator and a soft error attacked CNN accelerator as typical unstable accelerator examples.

Four convolution neural networks including LeNet, AlexNet, VGG-16 and VGG-19 are used. They are implemented on Xilinx KCU1500 based on 8bit fixed-point PipeCNN using SDAccel 2017.1. The FPGA cards is attached to a desktop computer configured with Intel(R) Core(TM) i7-2600 CPU (4core, 3.40GHz) via PCI-e 2.0. The communication bandwidth is 8GB/s.

A. Overclocked CNN accelerator

Clock frequency is almost proportional to the computation capability of the CNN accelerator when its architecture is determined. While timing analysis tools typically recommend a conservative clock frequency in order to avoid the possibility of timing violations, FPGA designs can be safely overclocked by a significant ratio with respect to the maximum operating frequency estimated by the FPGAs tool flow. This gives the advantage of increasing the implementation throughput without any design-level modifications. Beyond this safe overclocking margin, some critical paths in the design starts to fail and the output error rate increases exponentially with respect to the increase in the clock frequency. To tolerate the computing error and gain the performance benefit, we thus opt to use the proposed training framework.

With PipeCNN, we implemented four CNN including LeNet, AlexNet, VGG-16 and VGG-19 on KCU1500. As PipeCNN provides customized implementation for different CNN, the baseline frequency of the implementations is different. The frequency of the four implementations is 210Mhz, 210MHz, 190MHz and 190MHz respectively. Then we boost the clock frequency gradually and train for the unstable CNN accelerator implementations on ImageNet data set. The prediction accuracy of the resulting implementations is presented in Figure 6. In general, overclocking can enhance the clock frequency by 19% to 26%. While applying the off-line

TABLE I
HIGH-LEVEL INTERFACE TO CAFFE

ID	Function Name	
1	launchAccelerator()	It co
2	dataToFPGA(weight, input, wgtDevAddr, inDevAddr)	It transf
3	dataFromFPGA(outputDevAddr, output)	It transfers all the inter
4	convertIntToFloat(int iData, float fData)	It conver
5 and weight data to fixed point or integer for forward processing on the accelerator.	convertFloatToInt(float fData, int iData)	
6	dataLayoutReorder(data, reorderedData)	It reorders the data layout
7	dataLayoutRecover(reorderedData, data)	It reorders th

trained model to the accelerator with overclocking, the top-5 accuracy degrades by up to 4.3%. When the proposed training framework is used, the resulting retrained model can be much better especially near the overclocking limit.

For AlexNet, VGG-16 and VGG-19, the top-5 accuracy of the retained models is improved by 3.4%, 1.8%, and 2% respectively at the extreme overclocking frequency. For LeNet which is a rather small yet reliable network compared to the other three, the implementation remains unaffected even when the clock is boosted to 260 MHz from 210 MHz. When the clock goes up to 270MHz, the timing error can no longer be tolerated by the hardware system, the prediction accuracy drops to 10% which is essentially meaningless. In this case, the base model and the retrained model is pretty much the same. To ensure the stability of the overclocking experiment, we also keep measuring the accuracy of the accelerators with extreme overclocking. With repeatedly running the test for up to 40 hours, the measured accuracy varies slightly as present in Figure 7. Despite the fact that the errors caused by the overclocking can be hardly modeled precisely at runtime, the inherent error patterns may still partly be captured by the CNN model with the proposed training. This explains the higher prediction accuracy with the retrained model. According to the above experiments, we can conclude that the proposed accelerator aware training can produce more resilient CNN model tolerating errors caused by intensive overclocking.

Finally, we also present the training time on the hybrid CPUFPGA architecture. It can be seen that the training is much slower than the fixed-point training on CPU. This is mainly caused by the frequently data transferring between device memory and host memory in the proposed training, while this will not affect the inference time. In addition, we can also find that the training on larger network takes longer time and higher clock frequency is also beneficial to the training time as expected.

B. CNN accelerator with soft errors

With the shrinking semiconductor feature size and increasing FPGA capacity, FPGA design gets error-prone to the transient faults (often known as soft errors). They can affect the behaviors of the FPGA design dramatically. Many researchers [10]–[14] have proposed diverse approaches to address this problem. While CNN accelerators on FPGA can be different from general hardware design because the CNN model deployed can be further trained and tolerate the soft errors as proposed in prior section [15].

To explore the influence of soft errors on CNN accelerator, we need to inject soft errors to the system first. A number of fault injection techniques have been proposed in prior literature. In this work, we adopt a simple software simulation-based method to inject random errors. Although the error may be caused by on-chip memory or other SRAM cells, we have a random bit of the computing result flipped at a specific rate. The simple yet representative model will not increase the training time too much.

We also take LeNet, AlexNet, VGG-16, and VGG-19 to evaluate the influence of soft errors on prediction accuracy, the top-5 accuracy of the resulting implementations is presented in Figure 9. When we gradually increase the error rate from 1E-7 to 1E-5, the prediction accuracy degrades accordingly when applying the off-line trained model directly on the faulty accelerator. When the error rate goes up to 1E-4.5, the accuracy in the worst case drops by around 13.5%. Similar to overclocking, LeNet can tolerate more errors than the other three networks. The accuracy remains unchanged until the error rate reaches 1E-3. When the error injection rate is low, the CNN model is able to cover almost all the negative influence on the prediction accuracy.

When the error injection rate goes higher, the proposed retraining becomes critical. According to the experiments, the accuracy of the four retrained models improve by 6.8%, 1.5%, 3%, and 3% respectively compared to that of the base model when the accelerators are exposed to the highest error injection. In summary, the experiments demonstrate that we can have the CNN model to learn both the characteristics of the data and the underlying undeterministic behaviors of the accelerator together using the proposed training framework. The resulting CNN model can improve the accuracy without any modification on the accelerator when there is high error injection rate.

V. RELATED WORK

CNN accelerator: There have been notable efforts made to create hardware accelerators of machine learning algorithms for the sake of higher performance and energy-efficiency [16] in the past few years. Among the accelerators, the regular 2D array architecture has become a mainstream solution because of the relatively higher PE and bandwidth utility. Runtime reconfigurable PE arrays are applied to provide customized solutions for efficient CNN inference on FPGAs [4], [17]. In [18], an array of processing elements (PEs) with novel architecture was developed. With intensive data reuse, it reduces the external memory bandwidth requirements dramatically and outperforms the systolic-like structure proposed in [17]. Compared to the compact hardware design in [17], [18], Wei X et al. in [19] implemented a high-throughput CNN design and did comprehensive design space exploration on top of accurate models to determine the optimal design configuration.

Training of accelerators: Training approaches for CNN accelerator can be classified into three categories: (1) convert a pretrained floating point CNN model into a fixed point model without training, (2) train a CNN model with fixed point constraint, and (3) FPGA-implemented forward & backward propagation training tools. For first category, [20] applied codebook based on scalar and vector quantization methods in order to reduce the model size. [16] analyzed the quantization sensitivity of the network for each layer and then manually decide the quantization bit-widths. [21] find direct quantization for fixed-point network design does not yield good results and optimized the fixed-point design by employing back propagation based retraining. [9] adapted a higher precision for the parameters during the updates than during the forward and backward propagations for accumulating small changes in the parameters. [21] used only binary weights to train deep neural networks.

However, these approaches of the former two categories are not suitable for unstable circuit. For the third category, FCNN [22] reconfigured a streaming data path at runtime to cover the training cycle for the various layers in a CNN. Caffeine [17] provides tunable parameters, including the number and size of input/output feature maps, shape and strides of weight kernels, pooling size and stride, ReLU kernels, and the total number of CNN/DNN layers. Caffeinated FPGA [23] implemented FPGA kernels for forward and backward for Caffe and these kernels target the Xilinx SDAccel OpenCL environment for training and inference with CNNs. However, these approaches did not consider the unstable hardware behavior into their framework or either gave a way to train CNN under the undeterministic situation.

Unstable Hardware Behavior: Overclocking, soft Error, circuit defect induced by process variation etc. result in the undetermined behavior of the circuits. Overclocking, a technique to gain the additional performance from a given component by increasing its operating speed, may cause timing error. [7] gave the strands of research of arithmetic precision determination and overclocking. Razor [8] projected scaled the supply voltage and clock frequency beyond the most conservative value. Soft errors are unintended transitions of logic state in a circuit typically caused external source of ionizing radiations. The shrinking transistor sizes increased the soft-errors. [10] proposed An Automated SEU Fault-Injection Method and Tool for HDL-Based Design. [24] inject single-bit flips into the register-transfer level descriptions of floating-point ALUs.

VI. Conclusion

In this paper, we propose to take the CNN accelerators undeterministic behaviors into consideration at training and have the CNN model to learn the accelerators behaviors. To that end, we further build an open-sourced training system based on Caffe on a hybrid CPU-FPGA architecture. Then use the training system to deal with an overclocked CNN accelerator and an accelerator with soft errors. According to our experiments, the proposed training can improve the prediction accuracy of four CNN models up to 3.4% when the CNN accelerator is overclocked on the extreme situation. This method is also beneficial to the CNN accelerators with soft errors. In the case with most soft errors, it improves the prediction accuracy up to 6.8% and by 3.58% on average. The disadvantage is the much longer training time due to the frequent data transfer between host memory and device memory. This problem can be resolved when porting the system to closely coupled CPU-FPGA architectures with shared memory.

References

[1] D. Wang, K. Xu, and D. Jiang, "Pipecnn: An opencl-based open-source fpga accelerator for convolution neural networks," in *2017 International Conference on Field Programmable Technology (ICFPT)*, Dec 2017, pp. 279–282.

[2] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '15. New York, NY, USA: ACM, 2015, pp. 161–170. [Online]. Available: http://doi.acm.org/10.1145/2684746.2689060

[3] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang, and H. Yang, "Going deeper with embedded fpga platform for convolutional neural network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '16. New York, NY, USA: ACM, 2016, pp. 26–35. [Online]. Available: http://doi.acm.org/10.1145/2847263.2847265

[4] Y. Wang, J. Xu, Y. Han, H. Li, and X. Li, "Deepburning: Automatic generation of fpga-based learning accelerators for the neural network family," in *Proceedings of the 53rd Annual Design Automation Conference*, ser. DAC '16. New York, NY, USA: ACM, 2016, pp. 110:1–110:6. [Online]. Available: http://doi.acm.org/10.1145/2897937.2898003

[5] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello, "Hardware accelerated convolutional neural networks for synthetic vision systems," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, May 2010, pp. 257–260.

[6] H. Zeng, R. Chen, C. Zhang, and V. Prasanna, "A framework for generating high throughput cnn implementations on fpgas," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '18. New York, NY, USA: ACM, 2018, pp. 117–126. [Online]. Available: http://doi.acm.org/10.1145/3174243.3174265

[7] K. Shi, D. Boland, and G. A. Constantinides, "Accuracy-performance tradeoffs on an fpga through overclocking," in *2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines*, April 2013, pp. 29–36.

[8] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: A low-power pipeline based on circuit-level timing speculation," in *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 36. Washington, DC, USA: IEEE Computer Society, 2003, pp. 7–. [Online]. Available: http://dl.acm.org/citation.cfm?id=956417.956571

[9] M. Courbariaux, Y. Bengio, and J. David, "Low precision arithmetic for deep learning," *CoRR*, vol. abs/1412.7024, 2014. [Online]. Available: http://arxiv.org/abs/1412.7024

[10] W. Mansour and R. Velazco, "An automated seu fault-injection method and tool for hdl-based designs," *IEEE Transactions on Nuclear Science*, vol. 60, no. 4, pp. 2728–2733, Aug 2013.

[11] S. Karim, J. Harkin, L. McDaid, B. Gardiner, J. Liu, D. M. Halliday, A. M. Tyrrell, J. Timmis, A. G. Millard, and A. P. Johnson, "Fpga-based fault-injection and data acquisition of self-repairing spiking neural network hardware," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2018, pp. 1–5.

[12] T. S. Nidhin, A. Bhattacharyya, R. P. Behera, T. Jayanthi, and K. Velusamy, "Verification of fault tolerant techniques in finite state machines using simulation based fault injection targeted at fpgas for seu mitigation," in *2017 4th International Conference on Electronics and Communication Systems (ICECS)*, Feb 2017, pp. 153–157.

[13] O. Subasi, C.-K. Chang, M. Erez, and S. Krishnamoorthy, "Characterizing the impact of soft errors affecting floating-point alus using rtl-ievel fault injection," in *Proceedings of the 47th International Conference on Parallel Processing*, ser. ICPP 2018. New York, NY, USA: ACM, 2018, pp. 59:1–59:10. [Online]. Available: http://doi.acm.org/10.1145/3225058.3225089

[14] S. Rsch and B. Vogel-Heuser, "A light-weight fault injection approach to test automated production system plc software in industrial practice," *Control Engineering Practice*, vol. 58, pp. 12 – 23, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0967066116302143

[15] F. Tu, W. Wu, S. Yin, L. Liu, and S. Wei, "Rana: Towards efficient neural acceleration with refresh-optimized embedded dram," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ser. ISCA '18. Piscataway, NJ, USA: IEEE Press, 2018, pp. 340–352. [Online]. Available: https://doi.org/10.1109/ISCA.2018.00037

[16] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 1–13. [Online]. Available: https://doi.org/10.1109/ISCA.2016.11

[17] C. Zhang, G. Sun, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2018.

[18] U. Aydonat, S. O'Connell, D. Capalija, A. C. Ling, and G. R. Chiu, "An opencl^{TM}deep learning accelerator on arria 10," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. New York, NY, USA: ACM, 2017, pp. 55–64. [Online]. Available: http://doi.acm.org/10.1145/3020078.3021738

[19] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, "Automated systolic array architecture synthesis for high throughput cnn inference on fpgas," in *Proceedings of the 54th Annual Design Automation Conference 2017*, ser. DAC '17. New York, NY, USA: ACM, 2017, pp. 29:1–29:6. [Online]. Available: http://doi.acm.org/10.1145/3061639.3062207

[20] Y. Gong, L. Liu, M. Yang, and L. D. Bourdev, "Compressing deep convolutional networks using vector quantization," *CoRR*, vol. abs/1412.6115, 2014. [Online]. Available: http://arxiv.org/abs/1412.6115

[21] K. Hwang and W. Sung, "Fixed-point feedforward deep neural network design using weights +1, 0, and -1," in *2014 IEEE Workshop on Signal Processing Systems (SiPS)*, Oct 2014, pp. 1–6.

[22] W. Zhao, H. Fu, W. Luk, T. Yu, S. Wang, B. Feng, Y. Ma, and G. Yang, "F-cnn: An fpga-based framework for training convolutional neural networks," in *2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, July 2016, pp. 107–114.

[23] R. DiCecco, G. Lacey, J. Vasiljevic, P. Chow, G. Taylor, and S. Areibi, "Caffeinated fpgas: Fpga framework for convolutional neural networks," in *2016 International Conference on Field-Programmable Technology (FPT)*, Dec 2016, pp. 265–268.

[24] O. Subasi, C.-K. Chang, M. Erez, and S. Krishnamoorthy, "Characterizing the impact of soft errors affecting floating-point alus using rtl-ievel fault injection," in *Proceedings of the 47th International Conference on Parallel Processing*, ser. ICPP 2018. New York, NY, USA: ACM, 2018, pp. 59:1–59:10. [Online]. Available: http://doi.acm.org/10.1145/3225058.3225089