CONCRETE: A PER-LAYER CONFIGURABLE FRAMEWORK FOR EVALUATING DNN WITH APPROXIMATE OPERATORS

Zheyu Liu, Guihong Li, ⋆Fei Qiao, Qi Wei, Ping Jin, Xinjun Liu, Huazhong Yang

Tsinghua University
⋆Corresponding Author: qiaofei@tsinghua.edu.cn

ABSTRACT

Approximate computing has drawn considerable attention to both academia and industry in the area of DNN hardware. Despite substantial efforts to design approximate circuits and building blocks, the resilience of DNN layers and structures remains an untapped field to explore. This paper presents an efficient framework to evaluate DNN resilience with fine-grained approximate operations, such as multipliers, adders and low-bit operators. The framework can execute large-scale approximate DNNs with relatively less time overhead. Massive experiments are conducted with the proposed framework to reveal the relationship between network structures and error tolerance. Additionally, a case study of fine-tuning the approximate DNN is presented.

***Index Terms*—** Approximate Computing, Neural Networks, Low-power Design, Optimization Framework

1. INTRODUCTION

Nowadays, DNNs (Deep Neural Networks) have been extensively explored and deployed in various computing devices. From edges to clouds, performance and energy efficiency of DNN inference are two critical factors which have been studied and optimized in substantial works. A new trend in this field is to excavate the inherent fault resilience of DNNs. Combined with the imperfection of hardware, tradeoffs could be made between computing quality and hardware efficiency. This design methodology has become a promising paradigm in the post-Moore era, known as Approximate Computing. This paper presents an efficient framework to qualify the resilience of DNNs with approximate operators, providing insights for hardware-software co-design and optimization.

Unlike the near-perfect requirement in conventional computing systems, DNN inference only requires a "good-enough" calculation to guarantee reasonable application accuracy [1]. This relaxation of computing quality can enable significant improvement in hardware efficiency, exhibiting great opportunities in HW-SW co-design for inference engines. Motivated by this, various approximate building blocks and architectures are emerging, such as approximate multipliers [2, 3, 4, 5], adders [6, 7, 8], and memory blocks, as well as algorithm designs [9, 10], such as weight compression and quantization [11, 12, 13, 14]. These approximate techniques have different mechanisms of error occurrence. For instance, SRAM failure caused by the near-threshold operation and process variation yield a static error pattern which is data-independent [15], while errors produced by operators (e.g., multipliers and adders) are usually strongly correlated to input data and dynamically change in the computing procedures[2]. These kind of differences cause uncertainty in the design phase of DNN hardware, and the reliability issues after fabrication. Therefore, it is vital to qualify the resilience of DNNs for various approximate operators and techniques before proceeding to HW-SW co-design.

To maximize the hardware efficiency while providing quality guarantees, we need to thoroughly examine and understand the resilience of a DNN computing graph. In other words, we have to determine to what extent a DNN model could be approximated and where the criterion of a "good-enough" operator lies in. The DNN inference has long been treated as an end-to-end model in IoT applications. However, in the context of approximate computing, we need to look inside the black box because the resilience varies in layer and structure level. Such fine-grained analysis leads to a large design space to explore. Thus an efficient framework is essential to provide per-layer resilience evaluation before deploying the model to approximate hardware.

In this paper, we present a DNN resilience evaluation framework: Concrete. Concrete enables us to analyze per-layer and per-structure resilience to approximate multipliers/adders and fixed-point quantization (weight/activation compression). Conventional frameworks [15] only support static error simulation such as storage device failure, which is simple to implement with data preprocessing, yet not feasible for approximate operators which dynamically produce errors during the inference computing. Concrete is a fast and per-layer configurable framework that can explore design space constructed by different types of approximate operator model, such as LUT (Look Up Table) and logic behavioral model. With approximate kernel design in BLAS (Basic Linear Algebra Subprograms) level, Concrete can execute approximate operations on GPUs with a relatively small time overhead. This feature is vital for large-scale DNN models such as VGG and ResNet for running ImageNet classification. Besides, Concrete is independent of hardware architecture because the approximate operation is fine-grained to MAC (multiply-accumulate) level. Hence the proposed framework is reliable for a range of architectures and hardware designs for running DNNs.

This paper makes the following contributions:

- We design Concrete to be an efficient and flexible framework that can rapidly evaluate the resilience of various DNN structures and layers on GPUs.
- The proposed framework offers user-friendly interfaces to transform an exact DNN model to an approximate version and supports different operator description such as LUT and behavioral model.
- We execute a range of the most popular DNNs with three approximate operations, presenting per-layer and per-structure sensitivity evaluations. According to the results, we provide some options for DNN HW-SW co-design.
- The proposed framework supports re-training (fine-tuning) to enhance the resilience of DNN models further. A case study is provided to demonstrate this.

2. THE CONCRETE FRAMEWORK

2.1. An overview

An overview of the proposed framework is shown in Figure 1. Concrete is constructed by three fundamental building blocks: user interface (a parser based on Python), the inference engine and the fine-

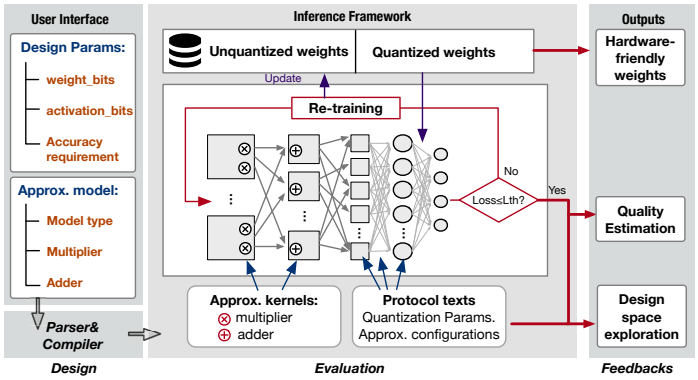

Fig. 1. An overview of the proposed framework: Concrete. This framework provides approximate inference evaluation and weights fine-tuning for approximate models.

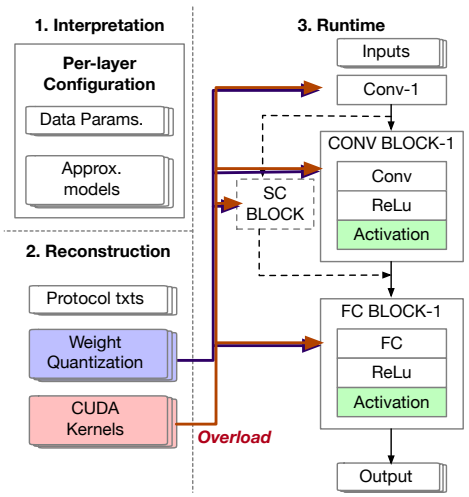

Fig. 2. Implementation of Concrete inference framework. Both static approximation and dynamic approximation are implemented, but happen in different procedures of the whole process.

tuning engine. The user interface is an interpretation program implemented by Python. It allows designers to define approximate design parameters and models using CAFFE-style protocol texts. Then interpret the texts into executable scripts and compile the operation models into CUDA kernels. The kernels are overridden into the inference framework to replace the exact operators in the inference engine according to user's configuration. In runtime, the reconstructed inference engine executes for requested iterations and generates the evaluation results and the approximate DNN models. The fine-tuning engine is inherited from CAFFE framework, in which the input of the fine-tuning program is re-designed to satisfy the Concrete output.

The Concrete framework provides useful feedbacks for HW-SW co-design procedure of DNN hardware. First, the framework can evaluate approximate inference model with a per-layer granularity, providing baselines for quality reconfigurable architecture design. Second, the framework could generate approximate hardware-friendly models by dynamic weight quantization and fine-tuning procedures. Last but not the least, the framework can be used to conduct design space exploration among diverse design choices. It benefits from the scalability of the Concrete framework which is capable of running various approximate operation models.

2.2. Concrete Inference Framework

Implementation of Concrete inference framework is illustrated in Figure 2. The overall inference program includes three stages: the interpretation stage, the reconstruction stage and the runtime stage. In the first stage, the parser interprets the input parameters and approximate model definitions into several executable scripts. Then the model definitions are compiled to generate CUDA kernels for running on GPUs. In the second stage, the main framework runs over all layers and executes the corresponding scripts to assign optional parameters such as fixed-point bit width and approximate factors. At the same time, the CUDA kernels are overloaded into the DLLs (Dynamic Link Libraries) to replace the original exact operators, reconstructing a user-defined approximate inference graph. Finally, in the runtime, the reconstructed graph is executed on GPUs to evaluate the model and resilience.

In general, there are two types of approximate operations respectively enrolled in two stages. The weight quantization, which is identified as the static approximation, happens in the reconstruction stage, while the approximate MAC operations(kernels) and the activation quantization perform in runtime, due to their strong dependence on input data. It is for such distinctions that Concrete could run in high speed because the weight quantization is considerable time-consuming if executed in runtime. Moreover, the approximate kernels are optimized to run using multithreading on GPUs, which further enhances the performance of Concrete. Time inspection for Concrete suggests that running a DNN inference (using ResNet-18 as an example in this context) with approximate operators only consume **1.05X-1.52X** more time than running the original software model. Moreover, Concrete running on GPU is **40.5X faster than executing on CPU**. In fact, it is quite time-consuming to use RTL-level simulation tools such as ModelSim to perform system level evaluation (usually in days even months). Yet with the help of the proposed Concrete framework, we could spend much less time (in minutes or hours) on the evaluation of the approximate neural network design before proceeding to actual hardware implementation.

Specifically, the parser for the user interface is built by Python, and the main framework is built based on CAFFE with C++ and the CUDA library. The overall implementation of Concrete inference process is summarized as the following pseudo code.

2.3. Benchmarks and Baselines

To demonstrate the efficiency and feasibility of the proposed framework, we collect five diverse DNN models to execute on Concrete, such as 3-layer MLP [16], LeNet-5 [17], CifarNet [18], VGG-16 [19], and ResNet-18 [20]. The dataset used for these DNN models are typical image classification tasks including MNIST, CIFAR10 and ImageNet. A brief overview of the benchmarks is presented in Table 1. For briefness of the large-scale DNNs such as VGG and ResNet, we cluster the similar layers and structures into blocks, i.e., CB stands for CONV-layer blocks, FB stands for FC-layer blocks, and SC stands for shortcut layer in ResNet. The approximate operations we implement in Concrete include weight/activation bits compression, approximate multiply and accumulate. These operations are modeled in different levels, and the diverse design options for these operations construct various design space. An overview of the approximate operations is presented in Table 2. All of the following evaluations are conducted by Concrete running on a server with a 3.6GHz Intel Core i7-6850k CPU and a single NVIDIA 1080ti GPU card.

Algorithm 1 Implementation of the Concrete Framework.

Input: Pre-trained model; Quantization params; Operator definitions.
Output: Classification accuracy.

```
procedure INTERPRETATION
    Parameter parsing;
    Approximate model compiling;
    Generate executable scripts;
end procedure
procedure RECONSTRUCTION
    for i in range (num. of layers) do
        if layer[i].approx() is true then
            Assign the approximate parameters;
            layer[i].weight− > quantization();
            Overload approximate kernels;
        end if
    end for
end procedure
procedure RUNTIME
    GPUThreads.init();
    for i in range (num. of iterations) do
        net.Inference− > run();
    end for
end procedure
```

Table 1. DNN benchmarks

Networks	Dataset	Structures	Baseline Accu.(fp32)
MLP	MNIST	FC-1,2,3	0.971
LeNet-5	MNIST	Conv-1,2 FC-1,2,3	0.989
CifarNet	CIFAR10	Conv-1,2,3 FB	0.759
VGG-16	ImageNet	CB-1,2,3,4,5 FB	0.886
ResNet-18	ImageNet	Conv-1 CB-1,2,3,4 (SC) FC	0.874

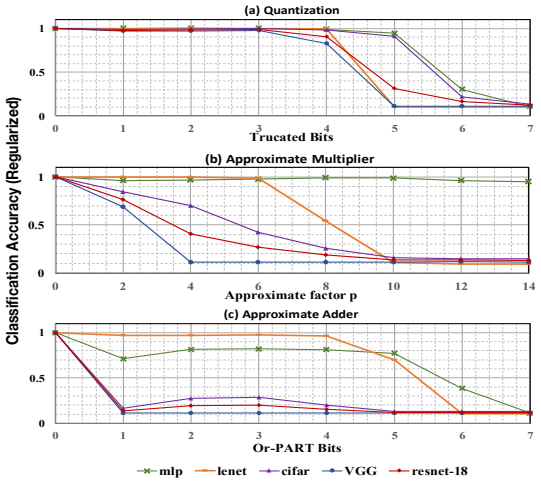

Fig. 3. An overall resilience evaluation of benchmarks using the approximate techniques in Table 2. All evaluations are conducted by Concrete running on GPU.

3. EVALUATIONS

3.1. Overall Network Resilience

The overall resilience evaluations for benchmarks are presented in Figure 3. The results exhibit a diversity of the fault tolerance of

Table 2. Approximate operations used in Concrete

Approx. Operator	Model	Option	Design Space
Multiplier [2]	LUT	Approx. factor	[2, 4, ..., 14]
Adder [6]	Logic	Or-bits width	[1,2,...,14]
Bit-width Compression	Behavioral	Bit width	[1,2,...,10]

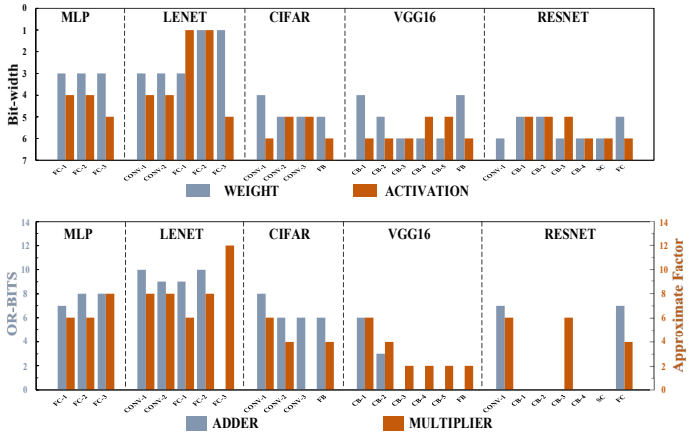

Fig. 4. Per-layer resilience qualification given by Concrete. Each bar corresponds to the highest approximate degree at which no accuracy loss is observed. Some layers are clustered as layer blocks to avoid verbosity.

DNNs with various approximate techniques. It appears that the DNNs have better tolerance to quantization error than to computing faults introduced by approximate multipliers and adders. The large-scale networks exhibit particularly lower tolerance to approximate operators due to considerably long accumulation paths. To better understand the resilience of each layer and structure of DNNs, we have to finer the granularity of evaluation with the help of the proposed Concrete framework. The fine-grained evaluation results are presented and analyzed in the following sections.

3.2. Per-layer Resilience Evaluations

Figure 4 presents the per-layer evaluation results acquired from Concrete. Each bar corresponds to the highest approximate degree at which no classification accuracy loss is observed compared with baselines. The results intuitively show that each layer of DNNs has a diverse resilience to approximate operations. Besides, the resilience significantly varies across the scale of DNNs and datasets. Such per-layer analysis provides opportunities to benefit quality-configurable architecture and hardware design.

The influence of quantization: The resilience to weights and activations is presented respectively in the upper subgraph of Figure 4. It shows that in most cases the intermediate layers exhibit similar resilience to both weights and activations. However, this conclusion is incorrect for the first and the last layer of DNNs. The results consistently reveal that the first and last layers of DNNs show higher resilience to weight quantization than activation quantization. This fact is comprehensible because the influence of information loss in the very first layer propagates across the entire network, causing lower-robustness feature extraction in the subsequent layers. Moreover, the output of the last layer directly corresponds to the classification re-

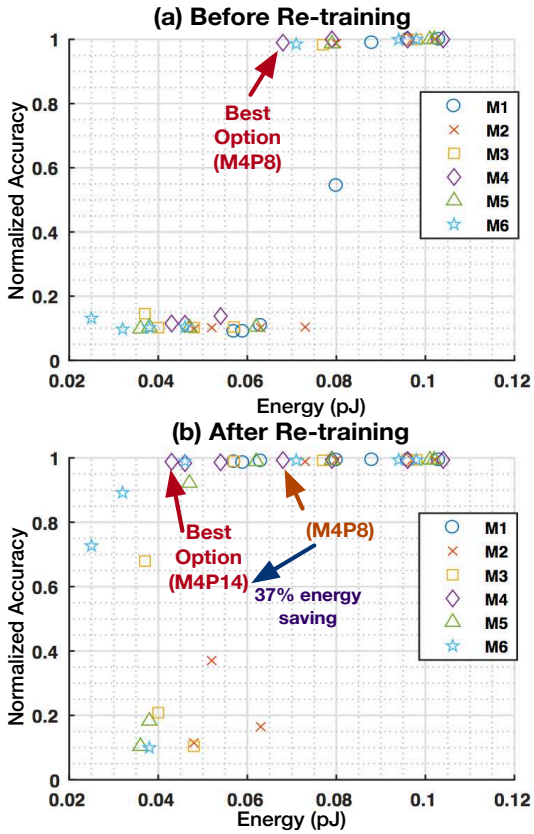

Fig. 5. A case study: design space exploration for approximate multipliers using in LeNet-5 hardware implementation.

sults. Changes are high that low-bit fixed-point quantization for the last layer finally results in misclassifications. This phenomenon suggests that, in actual fixed-point computing design, the first and last layer of DNNs should use higher quantization precision, especially for activation representation.

Fault tolerance to approximate operators: The lower subgraph in Figure 4 presents the relationship between operator approximation and layer resilience. The results show that the intermediate layers and structures in large-scale DNNs such as VGG and ResNet are considerably less tolerant to the approximate adders and multipliers. These layers have relatively dense connectivity and a large number of weights, forming sophisticated fault accumulation paths. As a consequence, the accumulated faults significantly change the distribution of intermediate data, causing lower application quality. The results suggest that the designers should be conscientious when applying approximate logics to such dense layers.

Resilience versus complexity: All the evaluation results reveal a possible relation between resilience and the complexity of DNNs and datasets. It shows that the smaller DNNs targeting to simpler datasets are usually much more tolerant to both data quantization and imprecise operators than large-scale DNNs and classification tasks. In actual approximate hardware design for DNNs, such fine-grained qualification is necessary to conduct in HW-SW co-design phase to ensure the feasibility of DNN chips after fabrication.

3.3. Fine-tuning for Better Resilience: A Case Study

A typical scenario of approximate hardware design is determining optimal design parameters (e.g., approximate factor) among numerous choices. Taking approximate multiplier [2] for example, two design parameters - M and P, denoting different approximate logic and approximate degree respectively - need to be specified. There are six kinds of logic (M1-M6) and seven approximation degrees (P2-P14) in total, formulating a design space in which 6x7=42 subtypes of the multiplier need to be explored. It is an impossible task to establish and simulate RTL descriptions for all these design options using traditional hardware design toolchains. Fortunately, with the help of the proposed framework, designers can explore the design space much more efficiently, and conduct fine-tuning to enhance further the resilience of the pre-trained models to gain higher hardware efficiency.

In this section, we present a case study of LeNet-5 leveraging 42 types of approximate multipliers. The results (see Figure 5) show the optimal multiplier for LeNet-5 is M4P8 for its lossless classification accuracy and relatively low energy consumption. Note that the result changes to M4P14 after fine-tuning the weights with the approximate inference engine. This change implies a significant improvement of the resilience of LeNet-5, and 37% more energy saving can be expected in actual hardware implementation. Table 3 presents the variation of classification accuracy after weight fine-tuning, demonstrating the importance of fine-tuning in the context of approximate hardware design for DNNs. The whole exploration is conducted by the proposed Concrete framework on a single NVIDIA 1080ti GPU card. The total time consumption for running 42 subtypes is less than 1 hour, of which 95% percent is consumed by the fine-tuning procedure.

Table 3. Quality variation of each design choice after fine-tuning.

gain(%)	P2	P4	P6	P8	P10	P12	P14
M1	-0.62	-0.64	0.45	44.39	**87.17**	**88.51**	**88.76**
M2	-0.64	-0.60	0.59	**87.49**	6.21	26.53	1.67
M3	-0.60	-0.55	0.90	**87.59**	0.19	10.52	52.80
M4	-0.62	-0.64	-0.58	0.32	**83.86**	**86.04**	**86.34**
M5	-0.64	-0.60	0.60	**87.53**	**81.07**	7.79	0.54
M6	-0.60	-0.55	0.69	**87.57**	-0.27	78.59	58.90

4. CONCLUSION

We present a fast and flexible framework, namely Concrete, that can rapidly execute and evaluate approximate DNN models, providing fine-grained resilience analysis to support HW-SW co-design for DNN inference engines. To demonstrate the efficiency and feasibility of the proposed framework, multiple popular DNNs ranging from MLP to VGG16 and ResNet are qualified with various approximate operators and models. The results exhibit some viewpoints and observations in the resilience aspect of DNNs. These findings and contributions of this paper could provide better options and insights for future approximate design for DNN hardware communities. The Concrete framework is architecture independent for now, which means Concrete only enables software-level evaluation. In our future work, we will take popular hardware NN architectures into account to support architecture-level approximate design evaluation.

5. ACKNOWLEDGEMENT

The authors would like to acknowledge support from National Natural Science Foundation of China under grant No. 91648116, 41871245, and National Key R&D Program of China under grant No. 2017YFC1500601, 2017YFB1400303. The authors would also acknowledge support from Beijing Innovation Center for Future Chip.

6. REFERENCES

[1] Zidong Du, Avinash Lingamneni, Yunji Chen, Krishna Palem, Olivier Temam, and Chengyong Wu. Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators. In *Proc. of 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 201–206, 2014.

[2] Weiqiang Liu, Liangyu Qian, Chenghua Wang, Honglan Jiang, Jie Han, and Fabrizio Lombardi. Design of approximate radix-4 Booth multipliers for error-tolerant computing. *IEEE Transactions on Computers (TC)*, 2017.

[3] Amir Momeni, Jie Han, Paolo Montuschi, and Fabrizio Lombardi. Design and analysis of approximate compressors for multiplication. *IEEE Transactions on Computers (TC)*, 64(4):984–994, 2015.

[4] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina. Evoapproxsb: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 258–261, March 2017.

[5] V. Mrazek, S. S. Sarwar, L. Sekanina, Z. Vasicek, and K. Roy. Design of power-efficient approximate multipliers for approximate artificial neural networks. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–7, Nov 2016.

[6] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas. Bio-inspired imprecise computational blocks for efficient vlsi implementation of soft-computing applications. *IEEE Transactions on Circuits and Systems I: Regular Papers (TCAS-I)*, 57(4):850–862, April 2010.

[7] Andrew B Kahng and Seokhyeong Kang. Accuracy-configurable adder for approximate arithmetic designs. In *Proceedings of the 49th Annual Design Automation Conference (DAC)*, pages 820–825, 2012.

[8] Yongtae Kim, Yong Zhang, and Peng Li. An energy efficient approximate adder with carry skip for error resilient neuromorphic VLSI systems. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, pages 130–137, 2013.

[9] Qian Zhang, Ting Wang, Ye Tian, Feng Yuan, and Qiang Xu. ApproxANN: an approximate computing framework for artificial neural network. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference (DATE) & Exhibition*, pages 701–706, 2015.

[10] S. S. Sarwar, S. Venkataramani, A. Raghunathan, and K. Roy. Multiplier-less artificial neurons exploiting error resiliency for energy-efficient neural computing. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 145–150, March 2016.

[11] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless CNNs with low-precision weights. *arXiv preprint arXiv:1702.03044*, 2017.

[12] Paul Merolla, Rathinakumar Appuswamy, John Arthur, Steve K Esser, and Dharmendra Modha. Deep neural networks are robust to weight binarization and other non-linear distortions. *arXiv preprint arXiv:1606.01981*, 2016.

[13] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1135–1143, 2015.

[14] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[15] Reagen Brandon, Gupta Udit, Pentecost Lillian, Whatmough Paul, Lee Sae Kye, Mulholland Niamh, Brooks David, and Wei Gu-Yeon. Ares: A framework for quantifying the resilience of deep neural networks. In *Design Automation Conference (DAC)*, 2018.

[16] Simon S Haykin. *Neural networks and learning machines*, volume 3. 2009.

[17] 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.

[18] Alex Krizhevsky. Alex's CIFAR-10 tutorial. `http://caffe.berkeleyvision.org/gathered/examples/cifar10.html`.

[19] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

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