# Efficient and Secure Deep Learning Inference System

## A Software and Hardware Co-design Perspective

by

Zhezhi He

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved July 2020 by the
Graduate Supervisory Committee:

Deliang Fan, Chair
Chaitali Chakrabarti
Yu Cao
Jae-sun Seo

ARIZONA STATE UNIVERSITY

August 2020

ABSTRACT

The advances of Deep Learning (DL) achieved recently have successfully demonstrated its great potential of surpassing or close to human-level performance across multiple domains. Consequently, there exists a rising demand to deploy state-of-the-art DL algorithms, e.g., Deep Neural Networks (DNN), in real-world applications to release labors from repetitive work. On the one hand, the impressive performance achieved by the DNN normally accompanies with the drawbacks of intensive memory and power usage due to enormous model size and high computation workload, which significantly hampers their deployment on the resource-limited cyber-physical systems or edge devices. Thus, the urgent demand for enhancing the inference efficiency of DNN has also great research interests across various communities. On the other hand, scientists and engineers still have insufficient knowledge about the principles of DNN which makes it mostly be treated as a black-box. Under such circumstance, DNN is like "the sword of Damocles" where its security or fault-tolerance capability is an essential concern which cannot be circumvented.

Motivated by the aforementioned concerns, this dissertation comprehensively investigates the emerging efficiency and security issues of DNNs, from both software and hardware design perspectives. From the efficiency perspective, as the foundation technique for efficient inference of target DNN, the model compression via quantization is elaborated. In order to maximize the inference performance boost, the deployment of quantized DNN on the revolutionary Computing-in-Memory based neural accelerator is presented in a cross-layer (device/circuit/system) fashion. From the security perspective, the well known adversarial attack is investigated spanning from its original input attack form (aka. Adversarial example generation) to its parameter attack variant.

*To my mother*

*To my father*

*In memory of my beloved paternal grandmother and maternal grandfather, who passed away during my graduate study*

ACKNOWLEDGMENTS

First and foremost, my deepest gratitude goes first to my committee chair, Prof. Deliang Fan, who expertly mentored me through my Ph.D. study and makes this dissertation possible. Back in 2015, Prof. Fan kindly offered me the opportunity to work with him, when I still have no clue of conducting research. Under his guidance, I got my first conference and journal publication in the first term right after I joined the research group. Later on, Prof. Fan has given me the incredible trust to formulate the research project based on my interests, and enlightened me to explore the challenging research topics. There were so many times, Prof. Fan and I work together all night long on revising paper. His persistent guidance leads to all the achievements I acquired during my Ph.D. study.

To my committee members, Prof. Chaitali Chakrabarti, Prof. Yu Cao, and Prof. Jae-sun Seo, I would like to express my heartfelt acknowledgment for all the suggestions and help in this dissertation, and the guidance to my future career path in academia. Besides, I appreciate all the help from my laboratory colleagues Shaahin Angizi, Jie Lin, Adnan Siraj Rakin, Li Yang, and Farhana Parveen. I would like to extend my sincere acknowledgments to all the personnel who give me encouragement, help, and guidance, which support me to achieve this milestone.

I gratefully acknowledge the funding support by nanoelectronics COmputing REsearch (nCORE) program that jointly funded by the national science foundation and semiconductor research corporation, and cyber Florida collaborative seed award program. The fellowships provided by the Arizona State University also financially support me in my last term of pursuing the Ph.D. degree.

Last but most importantly, I would like to express my greatest appreciation to my families and girlfriend for their unconditional love and support.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

Table Page

LIST OF FIGURES

# Chapter 1

# INTRODUCTION

## 1.1 Overview

In the last few years, deep learning technique (i.e., Deep Neural Network, DNN) [1] has achieved great success in multiple domains, such as image recognition [2], object tracking/detection [3], [4], machine translation [5] and etc [6]–[8]. Meanwhile, the number of edge devices (e.g., personal computers, smartphones, tablets, Internet-of-Things, etc) is increasing rapidly, which might be doubled in the next 5 years [9]. Rather than performing the DNN inference on the cloud, executing DNN inference on the edge devices is becoming more and more preferable, as it could reduce the inference latency[1], avoid bandwidth competition, enhance user privacy, cut the long-term cloud computing, and etc [10], [11].

Taken the classic image classification [12] as an example, the DNNs with higher accuracy normally accompany the larger model size and higher computing workload correspondingly, as shown in Table 1. Meanwhile, due to the memory capacity constraint of on-chip cache (normally < 10 MegaByte) in the edge-device system, such a high-performance model has to reside in the off-chip main memory, i.e., Dynamic Random-Access-Memory (DRAM). Nevertheless, for the conventional Von-Neumann architecture, the long-distance data communication (i.e., DRAM access) is emerging as the computation bottleneck, which is known as the "memory wall" [13]. As depicted

[1]Edge computing is faster than the cloud computing, since the long-distance data communication between edge and cloud is avoided.

Table 1: Deep convolution neural networks for image classification tasks, using ImageNet dataset. FLOP is the abbreviation of floating-point operations. MB is the abbreviation of MegaByte.

| Model | Total Parameters | Parameter size (MB) | FLOP (Giga) | Accuracy Top-1 (%) | Accuracy Top-5 (%) |
|---|---|---|---|---|---|
| AlexNet [12] | 61,100,840 | 223.08 | 1.43 | 56.55 | 79.09 |
| VGG-16 [14] | 138,357,544 | 527.79 | 30.96 | 71.59 | 90.38 |
| ResNet-50 [2] | 25,557,032 | 97.49 | 8.21 | 76.15 | 92.87 |
| DenseNet-201 [15] | 7,978,856 | 30.44 | 8.7 | 77.20 | 93.57 |
| Inception-v3 [16] | 23,834,568 | 90.92 | 11.45 | 77.45 | 93.56 |
| Wide ResNet-101-2 [17] | 126,886,696 | 484.03 | 45.58 | 78.84 | 94.28 |
| ResNeXt-101-32x8d [18] | 88,791,336 | 338.71 | 32.93 | 79.31 | 94.53 |

Table 2: Operation energy in 45nm CMOS process [19]. The relative energy cost of the off-chip DRAM access is significantly higher than other operations.

| Operations | Energy (pJ) | Relative Cost |
|---|---|---|
| 32-bit integer addition | 0.1 | 1 |
| 32-bit floating-point addition | 0.9 | 9 |
| 32-bit integer multiplication | 3.1 | 31 |
| 32-bit floating-point multiplication | 3.7 | 37 |
| 32-bit SRAM Access | 5 | 50 |
| 32-bit DRAM Access | 640 | 6400 |

in Table 2, the off-chip DRAM access owns orders higher energy-cost, in comparison to the on-chip operations.

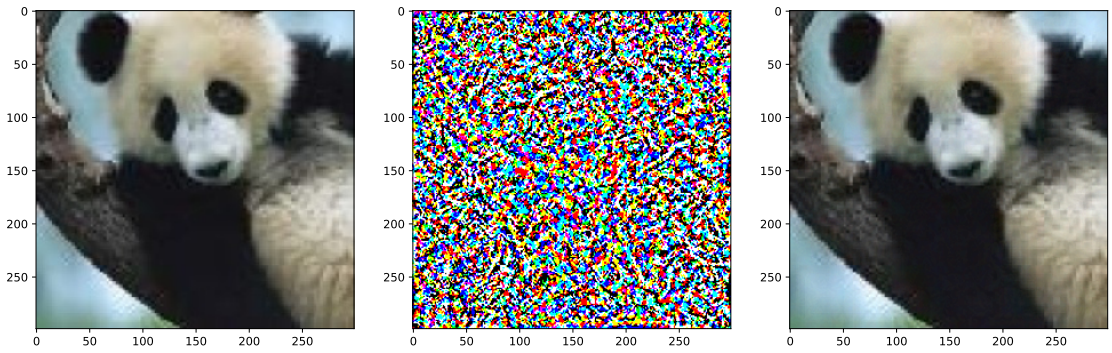
Taken the aforementioned concerns into consideration, the researchers and engineers have formed a general flow for DNN Deployment on the resource-constrained edge device, where the steps can be sequentially described:

1. Model compression: Given a target pretrained DNN, model compression is emerging as a must-have step to significantly shrink the size of model parameters

and computation complexity. The popular model compression techniques include quantization [20]–[22], pruning [23], [24], operator fusion [25], etc [26].

2. Model deployment: Thanks to the model compression techniques, massive DNN-friendly computing architectures are proposed to mainly address the memory wall issue of compressed neural network inference, with low-precision processing elements [27]–[29].

Due to the hardware platform (i.e., accelerator) that DNN running on is highly relying on the model compression scheme (e.g., model quantization), the neural network compression and deployment on the edge device could be viewed as a software and hardware co-design problem.



(a) Natural Data (b) Additive Noise (c) Adversarial Example

Figure 1: Illustration of adversarial example. The (a) natural image of giant panda [30] is applied with (b) additive noise to generate (c) adversarial example, then the giant panda in the adversarial example is recognized as indri by Inception-v3 [16]. Note that, the adversarial example is generated via Fast Gradient Sign Method (FGSM) [31].

Beyond the efforts for enhancing the efficiency of DNN inference discussed above, another perspective of deployed DNN should not be neglected is the security (or reliability in other words). Recent investigations [32]–[34] have gradually shown that DNNs are vulnerable to specially crafted adversarial attack (i.e., adversarial example).

The adversarial example is the DNN input maliciously perturbed by the human imperceptible noise, but can lead to the erroneous prediction of the target DNN, as shown in Fig. 1. Such vulnerability of DNN w.r.t the adversarial example has been demonstrated in many real-world scenarios as well, such as adversarial t-shirt [35] or tag [36] fooling the person detection surveillance system, and adversarial attack on road-signs (e.g., stop sign or speed limit sign) [37].

## 1.2 Statement of the Main Problems

Based on the discussion above, the current problems of the neural network running on hardware could be generally summarized into two categories: efficiency and security, which are specified as follows.

### 1.2.1 Efficiency

The model compression is an important step to empower the efficient inference to the target DNN running on hardware, where model quantization is one of the most popular techniques. Ideally, we are expecting a model quantization scheme could achieve significant model size reduction (i.e., weights compressed into lower bit-width representation), with negligible accuracy degradation. Such an objective has been pursued by many prior works [38]–[41], but drastic accuracy drop is still observed.

Beyond that, considering the hardware that DNN to be deployed, the conventional Von-Neumann architecture is countering the expensive long-distance data communication between processor and memory, which significantly hampers the computation efficiency. The In-Memory Computing (IMC) is an emerging computing paradigm con-

ducting low-precision computation within memory, thus avoiding the long-distance data access, as shown in Fig. 2. Thanks to the emerging resistive nonvolatile memory devices, for example, Resistive Random-Access-Memory (ReRAM), their utilization of constructing the IMC-based DNN accelerator can potentially enhance the speed-efficiency product by $3 \sim 4$ orders [42] compared to a custom digital ASIC. As the popular IMC-based dot-product engine, ReRAM crossbar [42], [43] could accelerate the dominant Multiplication-and-Accumulation (MAC) operations in the DNN inference, via leveraging its in-situ current-mode weighted summation operation. However, such an analog computing approach using ReRAM crossbar still counter great challenge in its real-world realization, due to various non-ideal effects [43].

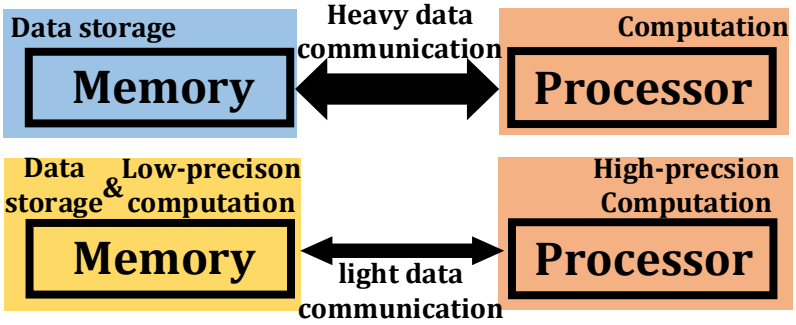



Figure 2: In-Memory Computing (IMC) to reduce the data communication and increase the computation parallelism.

### 1.2.2 Security

In terms of the neural network security, one of the main concerns is the vulnerability of DNN w.r.t the adversarial input attack (aka. adversarial example). There are various adversarial example generation methods proposed in prior works [32], [33], [44]–[47], while methods that can effectively enhance the resistance against adversarial example

are quite a few [33], [48]–[50]. By far, adversarial training [33] is still the most effective method to defend the adversarial attack. Therefore, it is an urgent demand to further push the state-of-the-art result of adversarial example resistance.

Furthermore, as the concept of adversarial attack is mainly focusing on attacking the natural input data fed to DNNs, the investigations of attacking the DNN parameters are not well explored yet. There exist prior works [51], [52] in an attempt to attack the model with full-precision parameters. However, our conducted simulation shows that randomly flipping the exponent part of floating-point weight could easily overwhelm the functionality of DNN [53]. Thus, more studies of adversarial attack upon DNN parameters are expected, in order to build reliable DNNs against adversarial attack variants.

## 1.3 Contributions

In this dissertation, we will focus on addressing several sub-problems under the framework specified above, where our contributions are highlighted in the following subsections.

### 1.3.1 Contribution 1: Optimized Neural Network Ternarization Scheme.

The first sub-problem we propose to address is the low bit-width neural network quantization scheme that leads to accuracy degradation. As a well-known low bit-width neural network quantization method, weight ternarization (-1, 0, +1) [41], [54], [55] is taken as a study case in this dissertation. The weight ternarization scheme not only reduces the model size by $16\times$ but also simplifies the dominant MAC operations into

addition and subtraction. However, many prior works [41], [54], [55] still show drastic accuracy degradation when weights are converted into ternary levels.

To mitigate such accuracy degradation, several optimized techniques are proposed by us [21], including iterative statistical weight ternarization and residual expansion methods. Using the image classification as benchmark, we test our iterative statistical weight ternarization method with AlexNet [12] and ResNet-18 [2] on ImageNet dataset [30], which both achieve the best top-1 accuracy compared to prior works [41], [54], [55]. If further incorporating our residual expansion method, compared to the full-precision counterpart, our ternarized ResNet-18 even improves the top-5 accuracy by 0.61% and merely degrades the top-1 accuracy only by 0.42% for ImageNet dataset, with 8$\times$ model compression rate.

### 1.3.2 Contribution 2: Countermeasures to Non-ideal Effects of ReRAM Crossbar

As we highlighted in the problem statement, the deployment of DNNs on the ReRAM crossbar based accelerator is hampered by various non-ideal effects, including Stuck-At-Fault (SAF) [56], [57], IR-drop [58], thermal noise, shot noise [59] and random telegraph noise [60], [61], etc. Prior works [62]–[65] have adopted different methods that all require retraining the target neural network's weight to be mapped in a crossbar array w.r.t various non-ideal effects for the specific device. However, the main drawback of such a method is that compute-intensive additional network retraining is required for each specific crossbar accelerator, which is not cost-efficient and not practical for future deep neural network deployment into different crossbar accelerators.

To examine the impacts of those non-ideal effects, we first develop a comprehensive framework called PytorX [43] based on main-stream DNN PyTorch framework [66].

PytorX could perform end-to-end training, mapping, and evaluation for a crossbar-based neural network accelerator, considering all the above discussed non-ideal effects of the ReRAM crossbar together. Our experiments based on PytorX show that directly mapping the trained large scale DNN into crossbar without considering these non-ideal effects could lead to a complete system malfunction (i.e., equal to random guess) when the neural network goes deeper and wider. In particular, to address SAF side effects, we propose a digital SAF error correction algorithm [21] to compensate for crossbar output errors, which only needs one-time profiling to achieve almost no system accuracy degradation. Then, to overcome IR drop effects, we propose a Noise Injection Adaption (NIA) methodology [43] by incorporating statistics of current shift caused by IR drop in each crossbar as stochastic noise to DNN training algorithm, which could efficiently regularize DNN model to make it intrinsically adaptive to non-ideal ReRAM crossbar. It is a one-time training method without the need for retraining for every specific crossbar. Optimizing system operating frequency could easily take care of rest non-ideal effects. Various experiments on different DNNs using image recognition applications are conducted to show the efficacy of our proposed methodology.

### 1.3.3 Contribution 3: Improve DNN Resistance Against Adversarial Example

Another contribution we made in this dissertation is to address the DNN vulnerability w.r.t the adversarial input attack (aka. adversarial attack). Taken the prediction accuracy without and with the adversarial attack as the evaluation metric (i.e., clean accuracy and attack accuracy respectively), adversarial training [33] is still the most effective method to defend the adversarial attack. Other defense methods [48]–[50] are achieving higher attack accuracy via the trade-off of clean accuracy.

Our solution is to utilize the regularization characteristic of noise injection to improve DNN's resistance against adversarial attacks. As the countermeasure against adversarial example, we propose Parametric-Noise-Injection (PNI) which involves trainable Gaussian noise injection at each layer on either activation or weights through solving the Min-Max optimization problem, embedded with adversarial training. These parameters are trained explicitly to achieve improved robustness. The extensive results show that our proposed PNI technique effectively improves the robustness against a variety of powerful white-box and black-box attacks such as PGD [33], C&W [45], FGSM [31], transferable attack [45], and ZOO attack [44]. Note that, PNI method improves both clean- and perturbed-data accuracy in comparison to the state-of-the-art defense methods, which outperforms current unbroken PGD defense by 1.1% and 6.8% on clean- and perturbed- test data respectively, using ResNet-20 architecture.

### 1.3.4 Contribution 4: Adversarial Attack on DNN Quantized Weight via Bit-flips and Its Defense

Last but not the least, our fourth contribution is to explore the new paradigm of the adversarial attack on neural network parameters. Instead of attacking DNN with extremely vulnerable full-precision weights as in [51], [52], our target is DNNs with quantized weights, whose weight magnitude is intrinsically constrained owing to the fixed-point representation.

To conduct an efficient bit-flip attack on weights, for the first time, we propose a Bit-Flip Attack (BFA) [53] together with Progressive Bit Search (PBS) technique, that can totally crush a fully functional quantized DNN and convert it to a random output generator with several bit-flips. With the aid of PBS, we can successfully at-

tack a ResNet-18 [2] fully malfunction (i.e., top-1 accuracy degrade from 69.8% to 0.1%) only through 13 bit-flips out of 93 million bits, while randomly flipping 100 bits merely degrades the accuracy by less than 1%. Beyond that, we conduct comprehensive investigations on BFA and propose to leverage binarization-aware training and its relaxation – piece-wise clustering as simple and effective countermeasures to BFA [67]. The experiments show that, for BFA to achieve the identical prediction accuracy degradation (e.g., below $11\%$ on CIFAR-10), it requires $19.3\times$ and $480.1\times$ more effective malicious bit-flips on ResNet-20 and VGG-11 respectively, compared to defend-free counterparts.

## 1.4 Organization

The organization of this dissertation is summarized as follows:

- In Chapter 2, the neural network quantization will be elaborated, which is the main technique in neural network compression and the basics for the neural network acceleration via in-memory computing (Chapter 3) and neural network weight security (Chapter 5). Moreover, an optimized weight ternarization scheme is discussed as a study case of neural network quantization.
- In Chapter 3, the ReRAM crossbar based in-memory computing accelerator is elaborated, where its various non-ideal effects are modeled and investigated. Those non-ideal effects are modeled as deterministic and stochastic noise which are considered during model training, called Noise Injection Adaption (NIA), to improve the fault-tolerant ability when adopted the crossbar based accelerator for model serving.

- In Chapter 4, one of the most important security concern of deployed neural network – adversarial input attack (aka. adversarial example) is discussed. A novel defense method called Parametric Noise Injection (PNI) is proposed to improve the DNN resistance against the adversarial input attack. The PNI can be considered as a trainable variant w.r.t the NIA discussed in chapter 3.
- In Chapter 5, we explore a brand-new paradigm of the adversarial attack on DNN weights, where the deployed DNN with quantized weights can be fooled by several bit-flips on the weight parameters. Such an attack is named as Bit-Flip Attack (BFA). Moreover, effective defense techniques are developed as well to improve the BFA resistance of target DNNs.

# Chapter 2

# NEURAL NETWORK MODEL COMPRESSION VIA QUANTIZATION

Neural Network (NN) compression has gradually emerged as a common technique to achieve the hardware-efficient inference of the target NN. The most popular NN compression techniques can be enumerated as quantization [20]–[22], [68], pruning [24], [69], [70], knowledge distillation [26], compact kernels [71]–[74], etc. Thereinto, neural network quantization has caught plentiful research interests and been widely discussed in literature [20]–[22], [39], [40], as it is a indispensable process of NN deployment on various edge devices (e.g., FPGA [28], [29], crossbar accelerator [43], [75]–[79]). This chapter provides an overview of the major neural network quantization techniques. Furthermore, an optimized weight ternarization scheme [21] is discussed as a study case of model quantization.

## 2.1 Basics of Neural Network Quantization

NN quantization normally refers to the process that constrains the inputs/weights/activation of the target NN from the continuous value to a discrete set (e.g., a codebook or integers). The main benefits resulting from the NN quantization can be summarized in twofold:

- Memory Usage Reduction. As the quantization converts the weights from the full-precision (i.e., 32-bit floating-point) [20] to multiple discrete levels, the weights can be encoded into a binary representation with reduced bit-width. It

could reduce memory usage for model storage, thus avoid expensive off-chip data access.

- Computation Simplification. With a specific quantizer, the dominant Multiply–ACcumulate (MAC) operation of NN inference can be significantly simplified. For example, the floating-point MAC can be simplified into a fixed-point counterpart with a uniform quantizer.

In the following subsections, the NN quantization will be elaborated from the perspectives of the quantizer (uniform/non-uniform), training scheme (post-training/training-aware), and quantization location (weight/activation).

Table 3: Overview of neural network quantization with various configuration.

| | | Accuracy degradation | Compression rate | Computation complexity | Training cost |
|---|---|---|---|---|---|
| Quantizer | Uniform | High | Low | Low | - |
| | non-uniform | Low | High | High | - |
| quantization scheme | post-training quantization | High | - | - | Low |
| | quantization-aware training | Low | - | - | High |

### 2.1.1 Quantizer

The quantizer design is not a brand-new topic, which was specified decades ago in the domain of signal processing [80]. Given a quantization function $Q(\cdot, \boldsymbol{t}, \boldsymbol{v})$, it partition a set of continuous value $\boldsymbol{x}$ ($\forall i, \boldsymbol{x}_i \in \mathbb{R}$) into quantized value $\boldsymbol{v}$, based on the quantization thresholds $\boldsymbol{t}$. Thus, the quantized version of $\boldsymbol{x}$ can be written as:

$$\hat{\boldsymbol{x}} = Q(\boldsymbol{x}, \boldsymbol{t}, \boldsymbol{v}); \quad \forall i, \hat{x}_i \in \boldsymbol{v} \tag{2.1}$$

which is illustrated in Fig. 3 To quantitatively measure the noise introduced by the quantization process, the mean-square error of $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$ (i.e. $n_e = \mathbb{E}(\boldsymbol{x} - \hat{\boldsymbol{x}})^2$) is normally adopted as the evaluation metric, which is known as quantization noise. In addition, the Signal-to-Quantization-Noise Ratio (SQNR) and Normalized Quantization Noise (NQN) can be expressed as:

$$\text{SQNR} = \frac{\mathbb{E}(\boldsymbol{x})^2}{n_e} = \frac{\mathbb{E}(\boldsymbol{x})^2}{\mathbb{E}(\boldsymbol{x} - \hat{\boldsymbol{x}})^2} = \frac{1}{\text{NQN}} \tag{2.2}$$

Thus, the design objective of quantizer is to select optimal quantization threshold and value for NQN minimization (i.e., SQNR maximization).

Depending on the distribution of data to be quantized, the quantizer can be generally divided into two corresponding categories: the uniform one and its non-uniform counterpart. Owing to the presence of regularization effect of weight decay (i.e., $L^2$ norm) and batch normalization [81] in the modern deep neural network, both the weights and intermediate feature map to be quantized are tending to follow the Gaussian-like distribution. Thus, we will focus on quantizing the non-uniform distributed data with both uniform and non-uniform quantizer in the following subsections.

#### 2.1.1.1 Uniform Quantizer

Quantization with uniform quantizer is commonly adopted for the data/signal following the uniform distribution (i.e., $\boldsymbol{x} \sim \mathcal{U}$) [80], for quantization noise minimization. The uniform quantization function can be written as [53]:

$$\begin{aligned} Q_{\text{u}}(x) &= \Delta x \cdot \text{round}(\text{clamp}(x)/\Delta x) \\ \text{clamp}(x) &= \max\Big( -\Delta x \cdot (2^{N_{\text{b}}-1} - 1), \min\big(x, +\Delta x \cdot (2^{N_{\text{b}}-1} - 1)\big)\Big) \end{aligned} \tag{2.3}$$

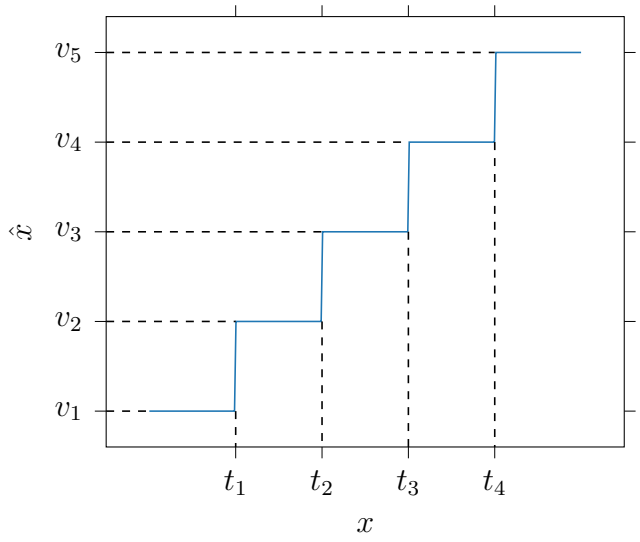



Figure 3: Quantization function with quantization threshold $\boldsymbol{t}$ and quantized value $\boldsymbol{v}$.

where $\Delta x$ denotes the quantization resolution chosen w.r.t $\boldsymbol{x}$. $N_\mathrm{b}$ is the bit-width used to encode the quantized values. clamp() is the clamping function with the upper-bound $+\Delta x \cdot (2^{N_\mathrm{b}-1} - 1)$ and lower-bound $-\Delta x \cdot (2^{N_\mathrm{b}-1} - 1)$. To obtain the quantization function $Q_\mathrm{u}$ with ideal quantization threshold and quantized values, the only parameter to be chosen is $\Delta x$:

$$\min_{\Delta x} \mathrm{NQN} \tag{2.4}$$

There are two methods can be leveraged for Eq. (2.4) optimization: exhaustive search and iterative projection, which are discussed as follows.

**Exhaustive Search.** The exhaustive search is a numerical method to choose $\Delta x$ for quantizer optimization. It uses the brute-force to examine all the candidates spanning from 0 to >max($\boldsymbol{x}$) in the small granularity of $\delta$, then select one out of it (i.e., $\Delta x \in \{i \cdot \delta\}, i \in \mathbb{Z}^+$). For better visualization, the relation between normalized quantization noise and clamping bound (i.e., $\Delta x \cdot (2^{N_\mathrm{b}-1} - 1)$) is illustrated in Fig. 4.

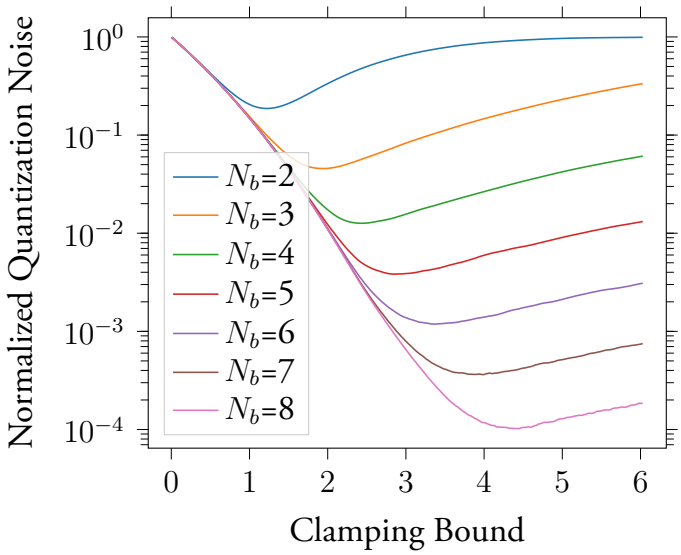



Figure 4: Normalized quantization noise versus clamping bound, for uniform quantizer w.r.t different quantization bit-width $N_\mathrm{b}$ configuration. The data to be quantized are 10000 samples from Gaussian distribution $\mathcal{N}(0,1)$ *i.i.d.*

**Iterative projection.** Another method for uniform quantizer optimization is leveraging the iterative projection [41], [82], which can be described by iteratively applying the following equations:

$$\text{Projection:} \quad \hat{\boldsymbol{x}}_\mathrm{int} = \frac{Q_\mathrm{u}(\boldsymbol{x})}{\Delta x} = \text{round}(\text{clamp}(\boldsymbol{x})/\Delta x) \tag{2.5}$$

$$\text{Update:} \quad \Delta x = \frac{\hat{\boldsymbol{x}}_\mathrm{int}^\mathrm{T} \cdot \boldsymbol{x}}{\hat{\boldsymbol{x}}_\mathrm{int}^\mathrm{T} \cdot \hat{\boldsymbol{x}}_\mathrm{int}} \tag{2.6}$$

Based on the description in [41], such iterative projection is convergence-guaranteed to the local minimum thanks to the decreasing loss in each iterative update. The quantization resolution can be initialized from $\Delta x = \max(|\boldsymbol{x}|)/(2^{N_\mathrm{b}-1}-1)$.

#### 2.1.1.2 Non-Uniform Quantization

To quantize the data of non-uniform distribution (e.g., Gaussian), the non-uniform quantizer normally can achieve lower quantization noise with smaller quantization bit-

width $N_{\mathrm{b}}$. The most well-known approach for non-uniform quantizer optimization is the Lloyd-Max method [80] (i.e., K-means), which is adopted in the famous deep compression in [20]. Since the non-uniform quantization is not the main quantization method adopted chapters, it is not specified here. However, a ternary weight quantization (i.e., low bit-width non-uniform quantization) scheme will be treated as a study case discussed in the next section.

### 2.1.2 Neural Network Quantization Scheme

#### 2.1.2.1 Quantization-aware Training

In this subsection, the quantization with uniform quantizer is mainly discussed.

##### 2.1.2.1.1 Gradient Approximation of Staircase Quantization Function

Almost for any quantization function which maps the continuous values into discrete space, it has encountered the same problem that such stair-case function is non-differentiable. Thus, a widely adopted countermeasure is using the so-called Straight-Through-Estimator (STE) [83] to manually assign an approximated gradient to the quantization function. We take the STE adopted in famous binarized neural network [84] as an example to perform the analysis, which is defined as:

$$\textbf{Forward}: \quad r_o = \mathrm{Sign}(r_i) \tag{2.7}$$

$$\textbf{Backward}: \quad \frac{\partial \mathcal{L}}{\partial r_o} \overset{\mathrm{STE}}{=} \frac{\partial \mathcal{L}}{\partial r_i}\bigg|_{|r_i|\leq 1} \Longrightarrow \frac{\partial r_o}{\partial r_i}\bigg|_{|r_i|\leq 1} = 1 \tag{2.8}$$

where $\mathcal{L}$ is the defined loss function. The rule behind such STE setup is that the output of quantization function $r_o$ can effectively represent the full-precision input

value $r_i$. Thus, Sign$(\cdot)$ performs the similar function as $f(r_i) = r_i$ whose derivative is $\partial f(r_i)/\partial r_i = 1$. However, the rough approximation in Eq. (2.7) and Eq. (2.8) leads to significant quantization error and hamper the network training when $r_i << 1$ or $r_i >> 1$. For example, given the Gaussian-sampled data to be quantized owning small variance (Var $<< 1$), applying the quantization in Eq. (2.7) will result in significant weight distribution shift which slows down the convergence speed.

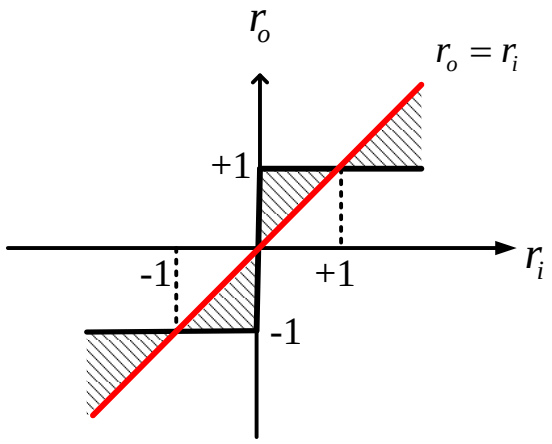



Figure 5: Analysis about the Straight-Through-Estimator (STE) design for $r_o = \text{Sign}(r_i)$. STE actually approximate the gradient of stair-case function (black) w.r.t the $r_o = r_i$ (red).

To encounter the drawback of naive STE design discussed above, a method called *gradient correctness* for better gradient approximation is proposed by us in [22]. For the uniform quantization function in Eq. (2.7), we follow the STE design rule which leads to the following expression:

$$\frac{\partial Q_u(x)}{\partial x} = \Delta x \cdot \frac{\partial \text{round}(\text{clamp}(x)/\Delta x)}{\partial x} \approx 1 \tag{2.9}$$

Thus, the STE for round$(\cdot)$ function can be derived as:

$$\frac{\partial \text{round}(\text{clamp}(x)/\Delta x)}{\partial x} \approx \frac{1}{\Delta x} \tag{2.10}$$

As depicted in Eq. (2.10), instead of simply assigning the gradient of round function as 1, we scale it w.r.t the value of $\Delta x$ on-the-fly. Such STE with gradient correctness could effectively improve the inference accuracy of weight-quantized DNN, which is supported by the experiments in [22].

#### 2.1.2.1.2 Quantization on Weight

Since the weight is stationary w.r.t the input, to quantize the weight, the weight quantizer is normally applied in a layer-wise fashion. For $l$-th layer ($l \in \{1, ..., L\}$), the quantization process from the floating-point base $\mathbf{W}_l^{\text{fp}}$ to its fixed-point (signed integer) counterpart $\mathbf{W}_l$ can be described as:

$$\Delta w_l = \max(\mathbf{W}_l^{\text{fp}})/(2^{N_q-1} - 1); \quad \mathbf{W}_l^{\text{fp}} \in \mathbb{R}^d \tag{2.11}$$

$$\mathbf{W}_l = Q_{\text{u}}(\mathbf{W}_l^{\text{fp}}) = \text{round}(\mathbf{W}_l^{\text{fp}}/\Delta w_l) \cdot \Delta w_l \tag{2.12}$$

Then, the quantized weights $\{\mathbf{W}_l\}_{l=1}^{L}$ is used to perform the DNN inference and the inference loss function $\mathcal{L}$ (e.g., cross-entropy loss) can be described as:

$$\mathcal{L}\Big(f(\{\mathbf{W}_l\}, \boldsymbol{x}), \boldsymbol{t}\Big) \tag{2.13}$$

were $\boldsymbol{x}$ and $\boldsymbol{t}$ are the vectorized input samples and the ground-truth labels respectively. To update the weight via back-propagation, the weight update is conducted upon the full-precision base, where its gradient can be computed via chain-rule:

$$\frac{\partial \mathcal{L}}{\partial w^{\text{fp}}} = \frac{\partial \mathcal{L}}{\partial w}\frac{\partial w}{\partial w^{\text{fp}}} \approx \frac{\partial \mathcal{L}}{\partial w}; \quad w^{\text{fp}} \in \{\mathbf{W}_l^{\text{fp}}\} \tag{2.14}$$

Then, the updated full-precision weight base is quantized using Eq. (2.12) and used for the inference loss computation, in the next training iteration.

#### 2.1.2.1.3 Quantization on Activation

In comparison to the weight quantization, applying the quantizer upon the activation (i.e., intermediate feature map) is always the difficult task, as the activation is dynamically varying w.r.t the DNN input samples. One of the most famous activation quantization methods [39], [85], [86] is the PArameterized Clipping acTivation (PACT) [85], which makes the quantizer clamping bound as a trainable parameter leveraging a modified straight-through-estimator. As the following chapters mainly utilize the weight quantization, details of activation quantization are not specified here.

### 2.1.2.2 Post-training Quantization

Owing to the NN training process is time-consuming (i.e., GPU hours) and the access to the entire training data is mandatory, post-training quantization is the most widely used NN quantization scheme in the industry as it is training-free. Such post-training quantization has been widely integrated into various deep learning frameworks, such as TensorRT[2], Pytorch[3], Tensorflow[4] and TVM[5].

For weight quantization, the quantizer is directly applied to the weight as in Eq. (2.12), but there is no training involved at all. For activation quantization in post-training quantization, the most popular approach is the entropy minimization

[2]https://docs.nvidia.com/deeplearning/frameworks/tf-trt-user-guide/index.html

[3]https://pytorch.org/docs/stable/quantization.html

[4]https://www.tensorflow.org/lite/performance/post_training_quantization

[5]https://tvm.apache.org/docs/tutorials/frontend/deploy_quantized.html

method introduced in [87]. Note that, although the idea of post-training quantization is to directly quantize the model without updating its parameters, we can still allow the statistical part of batch-normalization layer [81] to be updated by feeding several sample batches. Such a trick can improve $\sim 0.2\%$ top-1 prediction accuracy on the ImageNet dataset, based on the experiments conducted upon AlexNet and ResNet-18/34/50.

In the following sections, we will present an optimized weight ternarization method as a case study of neural network model compression.

## 2.2 Preliminaries of Low Bit-width Weight Quantization

Recently, model compression on the deep convolutional neural network has emerged as one hot topic in the hardware deployment of artificial intelligence. As the most popular technique, weight quantization techniques are widely explored in many related works which can significantly shrink the model size and reduce the computation complexity. Among all those works, DCNN with binary weight is the most discussed scheme, since it leads to $32\times$ model compression rate. More importantly, it also converts the original floating-point multiplication (i.e. *mul*) operations into addition/subtraction (i.e. *add/sub*), which could greatly reduce computational hardware resources and further dramatically lowers the existing barrier to deploy powerful deep neural network algorithm into low power resource-limited embedded system. BinaryConnect [38] is the first work of binary CNN which can get close to the state-of-the-art accuracy on CIFAR-10, whose most effective technique is to introduce the gradient clipping. After that, both BWN in [40] and DoreFa-Net [39] show better or close validation accuracy on ImageNet dataset. In order to reduce the computation com-

plexity to the bone, XNOR-Net [40] binarize the input tensor of the convolution layer which further converts the *add/sub* operations into bit-wise *xnor* and *bit-count* operations. Besides weight binarization, there are also recent works proposing to ternarize the weights of neural networks using trained scaling factors [54]. Leng et. al. employ the Alternating Direction Method of Multipliers (ADMM) to optimize neural network weights in configurable discrete levels to tradeoff between accuracy and model size [41]. ABC-Net in [88] proposes multiple parallel binary convolution layers to improve the network model capacity and accuracy, while maintaining binary kernel. All the above discussed aggressive neural network binarization or ternarization methodologies sacrifice the inference accuracy in comparison with the full precision counterpart to achieve large model compression rate and computation cost reduction.

## 2.3 Weight Ternarization Training Method

### 2.3.1 Ternarization With Iterative Statistical Scaling

As shown in Fig. 6, our training methodology to obtain accurate deep Convolutional Neural Network (CNN)[6] with ternarized weights can be divided into two main steps: Initialization and iterative weight ternarization training. We first train a designated neural network model with full precision weights to act as the initial model for the future ternarized neural network model. After model initialization, we retrain the ternarized model with iterative inference (including *statistical weight scaling*, weight

[6] In this work, all the convolution kernels and fully-connected layers do not have bias term, excluding the last layer.

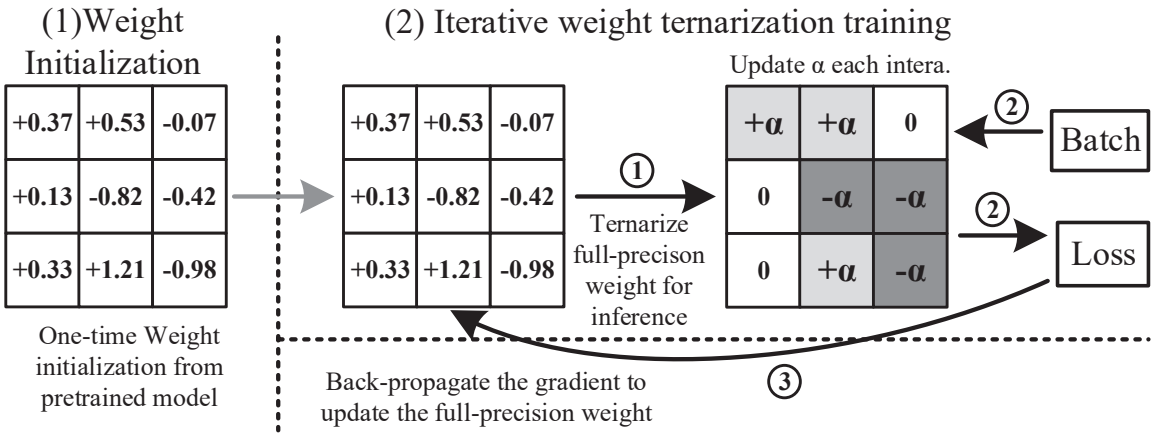



Figure 6: The training scheme of ternarized CNN model. (1)-(3) steps in the figure are iteratively operated during the training.

ternarization, and ternary weight based inference for computing loss function) and back-propagation to update full precision weights.

**Initialization:** The reason that we use a pre-trained model with full precision weights as the initial model comes in twofold: 1) Fine-tuning from the pre-trained model normally gives higher accuracy for the quantized neural network. 2) Our ternarized model with the initialized weights from the pre-trained model converges faster in comparison to ternarization training from scratch. For practical applications, engineers will use a full-precision model to estimate the ideal upper-bound accuracy. Afterward, various model compression techniques are applied to the full precision model to compress model size while trying to mitigate accuracy loss as well.

**Iterative weight ternarization training:** After loading the full-precision weights as initialization state, we start to fine-tune the ternarized CNN model. As described in Fig. 6, there are three iterative steps for the following training, namely, ①-statistical weight scaling and weight ternarization, ②-ternary weight based inference for loss function computation and ③-back propagation to update full precision weights.

① will first ternarize current full precision weights and compute the corresponding scaling factor based on current statistical distribution of full precision weight. For weight ternarization (i.e. -1, 0, +1), we adopt the variant staircase ternarization function, which compares the full precision weight with the symmetric threshold $\pm\Delta_{th}$ [55]. Such weight ternarization function in the forward path for inference can be mathematically described as:

$$\text{Forward}: \quad \boldsymbol{w}_l' = \begin{cases} \alpha \times Sign(w_{l,i}) & |w_{l,i}| \geq \Delta_{th} \\ 0 & |w_{l,i}| < \Delta_{th} \end{cases} \tag{2.15}$$

where $\boldsymbol{w}_l$ denotes the full precision weight tensor of layer $l$, $\boldsymbol{w}_l'$ is the weight after ternarization. We set the scaling factor as:

$$\alpha = E(|\boldsymbol{w}_{l,i}|), \quad \forall\{i \big| |\boldsymbol{w}_{l,i}| \geq \Delta_{th}\} \tag{2.16}$$

which statistically calculates the mean of absolute value of designated layer's full precision weights that are greater than the threshold $\Delta_{th}$. Unlike TWN uses $\Delta_{th} = 0.7 \times E(\boldsymbol{w}_l)$ as threshold, we set $\Delta_{th} = \beta \times \max(|\boldsymbol{w}_l|)$ as threshold [54], [55]. The reason is that, for each training iteration, the weight update causes large value drifting of $E(\boldsymbol{w}_l)$, which correspondingly leads to unstable $\Delta_{th}$. In this work, we employ single scaling factor for both positive and negative weights, since such symmetric weight scaling factor can be easily extracted and integrated into following Batch Normalization layer or ReLU activation function (i.e., both perform element-wise function on input tensor) for the hardware implementation.

Then, in ②, the input mini-batch takes the ternarized model for inference and calculates loss w.r.t targets. In this step, since all of the weights are in ternary values, all the dot-product operations in layer $l$ can be expressed as:

$$\boldsymbol{x}_l^T \cdot \boldsymbol{w}_l' = \boldsymbol{x}_l^T \cdot (\alpha \cdot Tern(\boldsymbol{w}_l)) = \alpha \cdot (\boldsymbol{x}_l^T \cdot Tern(\boldsymbol{w}_l)) \tag{2.17}$$

where $\boldsymbol{x}_l$ is the vectorized input of layer $l$. Since $Tern(\boldsymbol{w}_{l,i}) \in \{-1, 0, +1\}$, $\boldsymbol{x}_l^T \cdot Tern(\boldsymbol{w}_l)$ can be easily realized through addition/subtraction without multi-bit or floating point multiplier in the hardware, which greatly reduces the computational complexity.

In ③, the full precision weights will be updated during back-propagation. Then, the next iteration begins to re-compute weight scaling factor and ternarize weights as described in step-①. Meanwhile, since the ternarization function owns zero derivatives almost everywhere, which makes it impossible to calculate the gradient using chain rule in backward path. Thus, the Straight-Through Estimator (STE) [83][39] of such ternarization function in the back-propagation is applied to calculate the gradient as follow:

$$\text{Backward}: \quad \frac{\partial g}{\partial w} = \begin{cases} \frac{\partial g}{\partial w'} & if\ |w| \leq 1 \\ 0 & otherwise \end{cases} \tag{2.18}$$

where the gradient clipping prevents the full precision weight growing too large and stabilize the threshold $\beta \times \max(|\boldsymbol{w}_l|)$ during training (i.e., fine-tuning), thus leading to higher inference accuracy ultimately.

### 2.3.2 Residual Expansion to Improve Accuracy

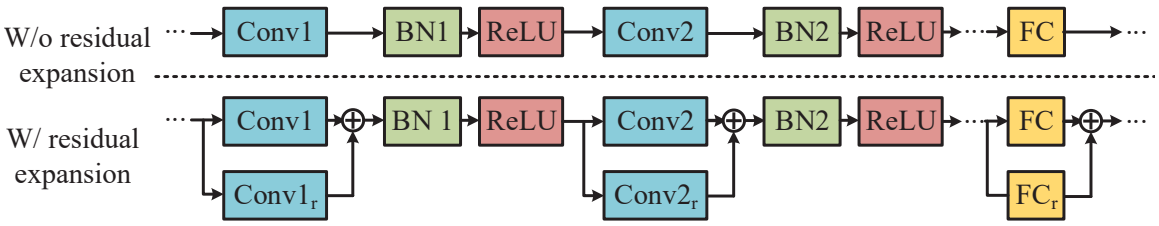



Figure 7: The block diagram of (top) original network topology and (bottom) the network with residual expansion to further compensate the accuracy degradation. In this figure, the expansion factor $T_{\text{ex}}$ is 2.

As the aforementioned discussion in the introduction, works like DoreFa-net [20], [39] adopt the multilevel quantization scheme to preserve the model capacity and avoid the accuracy degradation caused by extremely aggressive (i.e. binary/ternary) weight quantization. Since our main objective is to remove floating-point multiplication and only use addition/subtraction to implement the whole deep CNN, while minimizing accuracy degradation caused by weight ternarization. We propose a *Residual Expanded Layer* (REL) as the remediation method to reduce accuracy degradation. As depicted in Fig. 7, we append one REL over the convolution layers and fully-connected layers in the original network structure, where the expansion factor $T_{ex}$ is 2 in this case. We use *Conv* and *Conv$_r$* to denote the original layer and the added REL respectively. Both *Conv* and *Conv$_r$* are ternarized from the identical full precision parameters. During training, the network structure with RELs follow the same procedures as discussed in Fig. 6. The only difference is that *Conv* and *Conv$_r$* has two different threshold factors (e.g., $\beta = \{0.05, 0.1\}$ are in this work for $T_{ex} = 2$). We could append more RELs for compact CNN with low network redundancy.

The appended REL equivalently introduces more number of levels to each kernel. Assuming two threshold factors $a$, $b$ for *Conv* and *Conv$_r$* layers, where $a > b$, then we can formulate their computation as:

$$\boldsymbol{x}^T \cdot \boldsymbol{w}' + \boldsymbol{x}^T \cdot \boldsymbol{w}'_r = \boldsymbol{x}^T \cdot (\alpha \cdot \underset{\beta=a}{Tern}(\boldsymbol{w}) + \alpha_r \cdot \underset{\beta=b}{Tern}(\boldsymbol{w}_r)) \tag{2.19}$$

where the function $\alpha \cdot \underset{\beta=a}{Tern}(\boldsymbol{w}) + \alpha_r \cdot \underset{\beta=b}{Tern}(\boldsymbol{w}_r)$ is equivalent to a multi-threshold function that divides the element in $\boldsymbol{w}$ into levels of $\gamma = \{-\alpha - \alpha_r, -\alpha, 0, \alpha, \alpha + \alpha_r\}$, with thresholds $\{-b, -a, a, b\}$. Enlarging the expansion factor $T_{ex}$ will equivalently introduce more number of levels for the weights and recover the ternarized model capacity towards its full-precision baseline. However, compared to the case that directly use a

multi-bit kernels, the actual convolution kernel is still in ternary format and thus no multi-bit multiplication is needed (see more discussions in Section 2.5.3).

## 2.4 Experiments

In this work, all experiments are performed under the framework of Pytorch. The performance of our neural network ternarization methodology is examined using two datasets: CIFAR-10 and ImageNet. Other small datasets like MNIST and SVHN are not studied here since other related work [39], [55] can already obtain close or even no accuracy loss.

### 2.4.1 CIFAR-10

To impartially compare our ternarization method with [54], we choose the same ResNet-20, 32, 44, 50 (type-A parameter-free residual connection) [2] as the testing neural network architecture on CIFAR-10 dataset. CIFAR-10 contains 50 thousand training samples and 10 thousand test samples with $32 \times 32$ image size. The data augmentation method is identical as used in [2]. For fine-tuning, we set the initial learning rate as 0.1, which is scheduled to scale by 0.1 at epoch 80, 120, and 160 respectively. Note that, both the first and last layers are ternarized during the training and test stage.

we report the CIFAR-10 inference accuracy as listed in Table 4. It can be seen that only ternarized ResNet-20 gets minor (0.64%) accuracy degradation, while all the other deeper networks like ResNet-32, 44, 56 outperform the full precision baseline with 0.12%, 0.24%, and 0.18%, respectively. It shows that a more compact neural

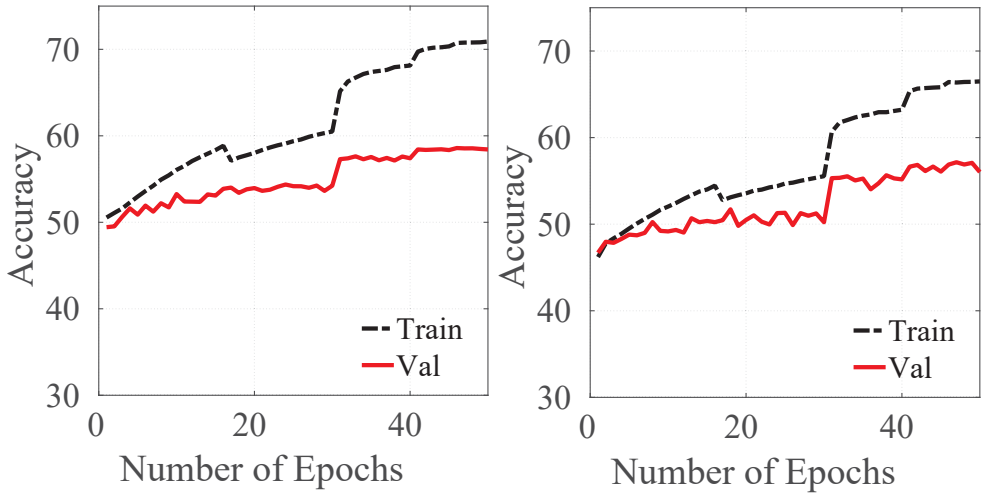
network, like ResNet20, is more likely to encounter accuracy loss, owing to the network capacity is hampered by this aggressive model compression. A similar trend is also reported in TTN [54]. Meanwhile, except for ResNet56, other accuracy improvements of our method are higher. Note that, the improvement is computed as the accuracy gap between pre-trained full precision model and corresponding ternarized model. Our pre-trained full precision model accuracy is slightly different even with the same configurations.

Table 4: Inference accuracy of ResNet on CIFAR-10 dataset

| | | ResNet20 | ResNet32 | ResNet44 | ResNet56 |
|---|---|---|---|---|---|
| TTN [54] | FP | 91.77 | 92.33 | 92.82 | 93.2 |
| | Tern | 91.13 | 92.37 | 92.98 | 93.56 |
| | Gap | -0.64 | +0.04 | +0.16 | +0.36 |
| our work | FP | 91.7 | 92.36 | 92.47 | 92.68 |
| | Tern | 91.65 | 92.48 | 92.71 | 92.86 |
| | Gap | -0.05 | +0.12 | +0.24 | +0.18 |

### 2.4.2 ImageNet

To provide more comprehensive experimental results on a large dataset, we examine our model ternarization techniques on image classification tasks with the ImageNet [30] (ILSVRC2012) dataset. ImageNet contains 1.2 million training images and 50 thousand validation images, which are labeled with 1000 categories. For the data pre-processing, we choose the scheme adopted by ResNet [2]. Augmentations applied to the training images can be sequentially enumerated as $224 \times 224$ randomly resized crop, random horizontal flip, pixel-wise normalization. All the reported classification accuracy on the validation dataset is a single-crop result. Beyond that, similar to




(a) First&last layers are Full-precision. (b) First&last layers are Ternarized.

Figure 8: Train and validation (top-1) accuracy of AlexNet on ImageNet

other popular model quantization works, such as XNOR-Net [40], DoreFa-Net [39], and TTN [54], we first keep the first and last layer in full precision (i.e., 32-bit float) and ternarize the remaining layers for a fair comparison. Besides that, we also conduct experiments to fully ternarize the whole network. Thus, in this case, the weight parameters of the whole network are in ternary format.

#### 2.4.2.1 AlexNet

We first present the experimental results and its analysis on AlexNet, which is similar to the variant AlexNet model used by DoreFa-Net [39] and TTN citezhu2016trained. The AlexNet in this work does not include Local-Response-Normalization (LRN) [12]. We apply a batch normalization layer after each convolution and fully-connected layer. The dropout layer is removed, since we find that it actually hampers the performance of fine-tuning of ternarized model from its full precision counterpart. A possible reason is

that the dropout method randomly deactivates some weighted connections. However, our kernel scaling factor is calculated with respect to the weight distribution of the entire layer.

We first train the full precision AlexNet with the aforementioned data augmentation method. SGD with momentum = 0.9 is taken as the optimizer. The initial learning rate is 0.1 which is scheduled with a decay rate of 0.1 at epoch 45 and 65. Batch size is set to 256, and weight decay is 1e-4. For fine-tuning ternarized AlexNet from the pre-trained full-precision model, we alter the optimizer to Adam. The initial learning rate is set to 1e-4 and scaled by 0.2, 0.2, and 0.5 at epoch 30, 40, and 45 respectively. The batch size is 256 as the full-precision model. Since low-bit quantization could also be considered as a strong regularization technique, weight decay is expected to be at a small value or even zero. Here we set the weight decay as 5e-6.

Table 5: Validation accuracy of AlexNet on ImageNet using various model quantization methods. FP, Bin., Tern. denote full-precision, binary and ternary respectively.

| | Quantize scheme | First layer | Last layer | Accuracy (top1/top5) | Comp. rate |
|---|---|---|---|---|---|
| DoreFa-Net[39], [54] | Bin. | FP | FP | 53.9/76.3 | ∼32× |
| BWN[40] | Bin. | FP | FP | 56.8/79.4 | ∼32× |
| ADMM[41] | Bin. | FP* | FP* | 57.0/79.7 | ∼32× |
| TWN[41], [55] | Tern. | FP | FP | 57.5/79.8 | ∼16× |
| ADMM[41] | Tern. | FP* | FP* | 58.2/80.6 | ∼16× |
| TTN[54] | Tern. | FP | FP | 57.5/79.7 | ∼16× |
| Full precision | - | FP | FP | 61.78/82.87 | 1× |
| this work | Tern. | FP | FP | 58.59/80.44 | ∼16× |
| this work | Tern. | Tern | Tern | 57.15/79.42 | ∼16× |

We here report the accuracy on ImageNet validation set including our work and other recently published works with state-of-the-art performance in Table 5. Note that,

we mark out the quantization state of the first layer and last layer for various methods. For works without such discussion in the paper and no released code, we consider that they configure the first and last layer in full precision (marked with *). We are confident in such an assumption since we double-checked the works they compared in their published papers are with first- and last-layer in full precision. The results show that our work achieves the best accuracy compared to the most recent works. In comparison to the previous best result reported by ADMM [41], our work improves the ADMM binary scheme and ternary scheme by 1.59% and 0.39% respectively.

Furthermore, different from previous works that all preserve the weights of the first and last layer in full-precision, we also conduct experiments to fully ternarize the whole network. Such ternarization leads to a further 1.44% top-1 accuracy drop. For a general-purpose processor, like CPU/GPU, the performance improvement by ternarizing the whole network is not quite huge since the parameter size reduction and computation cost reduction are relatively small. However, for a domain-specific neural network accelerator, like ASIC and FPGA, ternarizing the entire network means no specific convolution cores are needed to perform Multiplication and Accumulation (MAC) operations [89]. It will save a large amount of power, area, and other on-chip hardware resources, which further improve the system performance. We will provide a more quantitative analysis of this issue in the later discussion section.

#### 2.4.2.2 ResNet

Various works [54], [90] have drawn a similar conclusion that, under the circumstance of using identical model compression technique, accuracy degradation of the compressed model w.r.t its full-precision baseline will be enlarged when the model

topology becomes more compact. In other words, the neural network with a smaller model size is more likely to encounter accuracy degradation in weight compression. Meanwhile, since neural network with residual connections is the mainstream of deep neural network structure, we also employ the compact ResNet-18 (type-b residual connection in [2]) as another benchmark to demonstrate that our ternarization method could achieve the best state-of-the-art performance. The full-precision ResNet18 model [7] released by PyTorch framework is used as the baseline and pre-trained model in this work. The data augmentation method is the same as we introduced in Section 2.4.2.1 for AlexNet.

Table 6: Validation accuracy (top1/top5 %) of ResNet-18b [2] on ImageNet using various model quantization methods.

| | Quan. scheme | First layer | Last layer | Accuracy (top1/top5) | Comp. rate |
|---|---|---|---|---|---|
| BWN[40] | Bin. | FP | FP | 60.8/83.0 | ∼32× |
| ABC-Net[88] | Bin. | FP* | FP* | 68.3/87.9 | ∼6.4× |
| ADMM[41] | Bin. | FP* | FP* | 64.8/86.2 | ∼32× |
| TWN[41], [55] | Tern. | FP | FP | 61.8/84.2 | ∼16× |
| TTN[54] | Tern. | FP | FP | 66.6/87.2 | ∼16× |
| ADMM[41] | Tern. | FP* | FP* | 67.0/87.5 | ∼16× |
| Full precision | - | FP | FP | 69.75/89.07 | 1× |
| this work | Tern. | FP | FP | 67.95/88.0 | ∼16× |
| this work | Tern. | Tern | Tern | 66.01/86.78 | ∼16× |

As the simulation results listed in Table 6, ResNet-18b [2] with our ternarization scheme shows 0.95% and 0.5% higher top-1 and top-5 accuracy, respectively, compared to previous best result of ADMM [41] with equal compression rate (16×). ABC-Net [88] has $\sim 0.3\%$ higher top-1 accuracy over our work, but with a cost of 2.5 times

[7] https://github.com/pytorch/vision/blob/master/torchvision/models/resnet.py

larger model size. Moreover, our work saves ∼60% computation owing to the sparse kernel after ternarization. Similar as in AlexNet, if we further ternarize the first and last layer, both the top-1 and top-5 accuracy of ResNet-18 degrades.

Table 7: Inference accuracy (Top1/Top5 %) of ternarized ResNets on ImageNet dataset

| | ResNet18 | ResNet34 | ResNet50 | ResNet101 |
|---|---|---|---|---|
| FP | 69.75/89.07 | 73.31/91.42 | 76.13/92.86 | 77.37/93.55 |
| Tern | 66.01/86.78 | 70.95/89.89 | 74.00/91.77 | 75.63/92.49 |
| Gap | -3.74/-2.29 | -2.36/-1.53 | -2.13/-1.09 | -1.74/-1.06 |

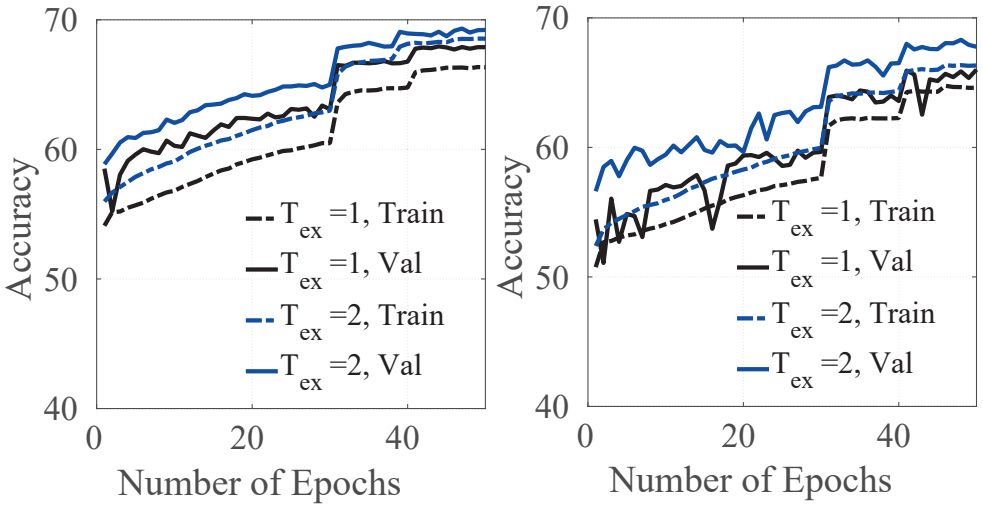
Moreover, we conduct a series of deeper residual networks (ResNet-34, 50, 101) in addition to the ResNet-18. As listed in Table 7, the accuracy gap between a ternarized model and its full-precision counterpart is reduced when the network goes deeper. Such observation is consistent with the experiments on the CIFAR-10 dataset.

#### 2.4.2.3 Improve Accuracy on ResNet With REL

We introduce the Residual Expansion Layer (REL) technique to compensate for the accuracy loss, which can still benefit from the weight ternarization in model size and computation reduction, and the multiplication elimination for convolution and fully-connected layers. In order to show that such residual expansion can effectively reduce the accuracy gap between the ternarized model and full-precision baseline, we report the validation accuracy of expanded residual ResNet18 on ImageNet in Table 8. When expansion factor $T_{ex}$ is 2 and two thresholds $\beta = \{0.05, 0.1\}$ are used, we succeeded to further shrink the accuracy gap to -1.7%/-1.03%. When the $T_{ex}$ is increased to 4 ($\beta = \{0.05, 0.1, 0.15, 0.2\}$ ), the accuracy gap is almost negligible.

Table 8: Validation accuracy (top1/top5 %) of ResNet-18b on ImageNet with/without residual expansion layer (REL).

| | First layer | Last layer | Accuracy (top1/top5) | Accuracy gap | Comp. rate |
|---|---|---|---|---|---|
| Full precision | FP | FP | 69.75/89.07 | -/- | 1× |
| $T_{ex}$=1 | FP | FP | 67.95/88.0 | -1.8/-1.0 | ∼ 16× |
| $T_{ex}$=1 | Tern | Tern | 66.01/86.78 | -3.74/-2.29 | ∼ 16× |
| $T_{ex}$=2 | FP | FP | **69.33/89.68** | **-0.42/+0.61** | ∼ 8× |
| $T_{ex}$=2 | Tern | Tern | 68.05/88.04 | -1.70/-1.03 | ∼ 8× |
| $T_{ex}$=4 | Tern | Tern | **69.44/88.91** | **-0.31/-0.16** | ∼ 4× |

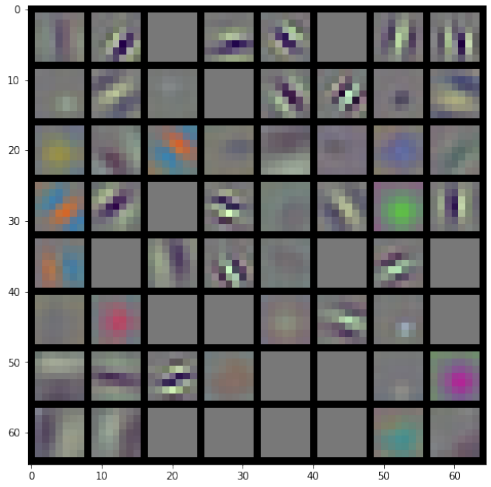



(a) First and last layer in full-precision (b) First and last layer are ternarized

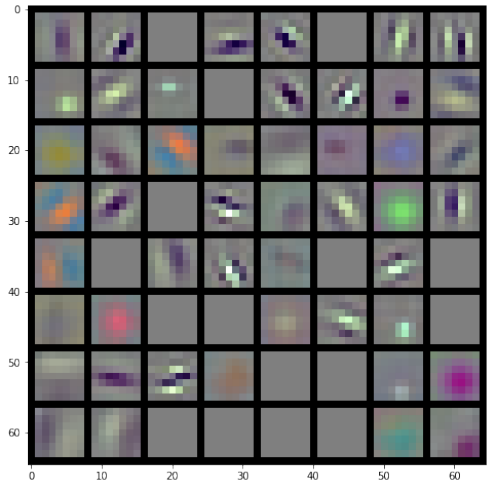
Figure 9: The accuracy evolution curve of ternarized ResNet-18b on ImageNet with residual expansion.

The training and test accuracy evolution accuracy of ResNet18 with all the configurations listed in Table 8 are plotted in Fig. 9. Based on our observation, whether implementing ternarization of the first and last layer does have a significant effect on the training process and accuracy. Ternarizing the first and last layer of ResNet18 (Section 2.4.2.3) causes large validation accuracy fluctuation during training compared to

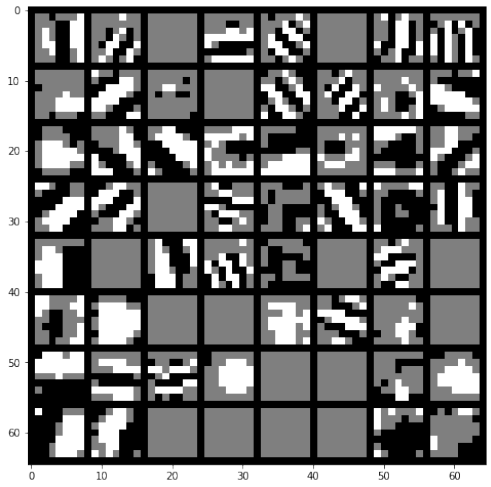
the experiments that keep the first and last layer in full precision (Section 2.4.2.3). As shown in Section 2.4.2.3, the validation accuracy curve with residual expansion alleviates the fluctuation and smooth the curve.



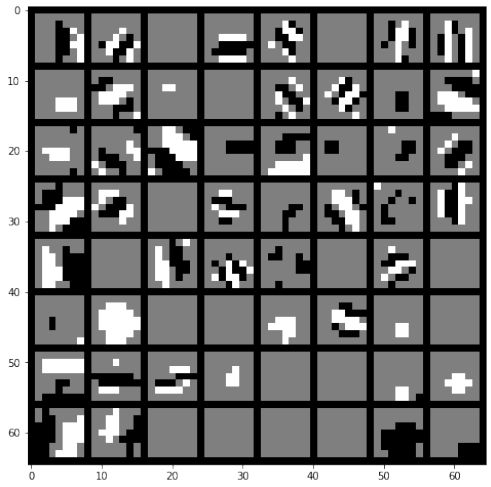
(a) Full-precision kernel before fine-tuning



(b) Full-precision kernel after fine-tuning



(c) Ternarized kernel with threshold factor $\beta = 0.05$.



(d) Ternarized kernel with threshold factor $\beta = 0.1$.

Figure 10: Kernel visualization for the first layer (channel 0) of ResNet18 before and after ternarization.

Beyond that, we further explore the effect of the ternarization process of the neural network with the assistance of kernel visualization. Since we also ternarize the first convolution layer of ResNet18, the input channel-0 is used as an example to study the weight transformation during weight ternarization as shown in Fig. 10. Kernels plotted in Section 2.4.2.3 are from the full-precision pre-trained model, which are taken as the initial weights. After fine-tuning the model for weight ternarization, the full-precision weights (i.e., the full precision weight is kept for back-propagation) shown in Section 2.4.2.3 still keep the similar patterns as in the pre-trained model. During inference in validation stage, kernels in Section 2.4.2.3 are ternarized with two different threshold factors, $\beta = 0.05$ and $\beta = 0.1$ for the original layer *Conv* and its REL $Conv_r$ respectively. As seen in Section 2.4.2.3 and Section 2.4.2.3, both ternarized kernels, to some extent, preserve the features from full precision model. Ternarized kernels with a larger threshold factor only keep the connection with higher weight intensity. Thus, the residual expansion layer with a higher threshold factor will lead to higher layer sparsity, and only a very small portion weights (i.e., non-zero value) will use the computing resources to perform network inference.

## 2.5 Ablation Study and Discussion

### 2.5.1 Ablation Study

To show the effectiveness of the proposed methods, we present a fair ablation study (Table 9) that isolates multiple factors which are listed as follow: 1) TW: directly training the Ternarized Weight (TW) from scratch without Iteratively Calculated Scaling factors (ICS); 2) TW+ICS: Training the ternarized network from scratch with the vary-

Table 9: Validation accuracy of ResNet-18b on ImageNet.

| | Accuracy | Comp. rate |
|---|---|---|
| Baseline: Full precision | 69.75/89.07 | 1× |
| TW | 57.48/81.15 | ∼16× |
| TW+ICS ($\beta = 0.05$) | 65.08/86.06 | ∼16× |
| TW+ICS ($\beta = 0.1$) | 64.78/86.06 | ∼16× |
| TW+ICS ($\beta = 0.2$) | 57.09/81.01 | ∼16× |
| TW+FT | 64.94/86.13 | ∼16× |
| TW+ICS+FT ($\beta = 0.05$) | 66.01/86.78 | ∼16× |
| TW+ICS+FT+REL ($\beta = \{0.05, 0.1\}$) | 68.05/88.04 | ∼8× |

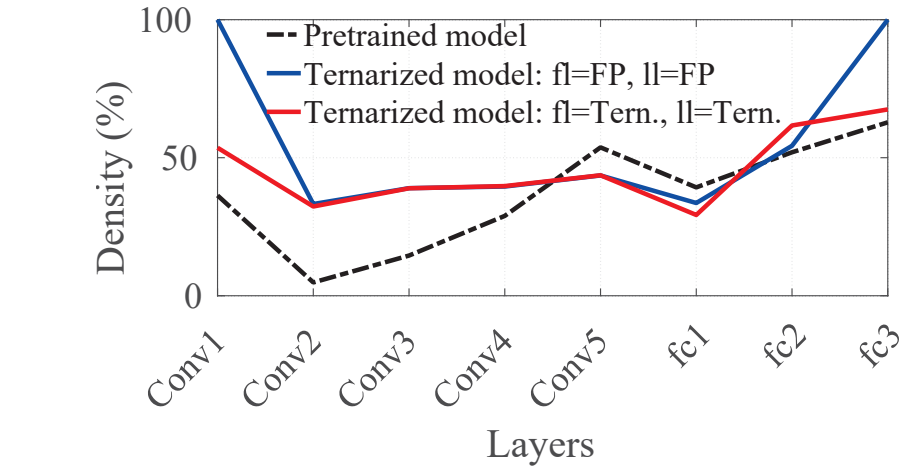
ing scaling factor ($\beta$=0.05, 0.1 and 0.2); 3) TW+FT: Fine-Tuning (FT) the ternarized weight. It shows that using ICS and training from scratch is close to the performance of fine-tuning the model without ICS. 4) TW+ICS+FT: Fine-tuning the model with ICS further improves the accuracy. 5) TW+ICS+FT+REL: incorporating our proposed REL techniques leads to almost no accuracy loss.

### 2.5.2 Deep CNN Density Examination

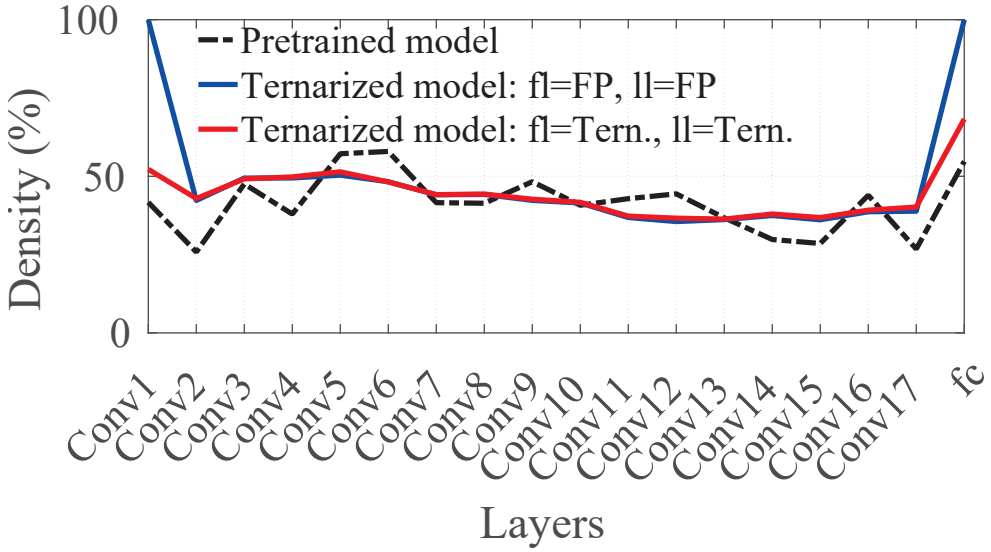
Table 10: Density variation of first and last layer before/after fine-tuning

| | First layer | | Last layer | | Average density |
|---|---|---|---|---|---|
| | Before | After | Before | After | |
| AlexNet | 36.3% | 53.6% | 62.8% | 67.5% | 41.1% |
| ResNet18 | 41.7% | 52.2% | 54.8% | 68.3% | 40.1% |

Let us assume the trained sparsity (sparsity=1-density) of the designated layer is correlated to its redundancy. Fig. 11 depicts the weight density distribution of all the convolution and fully-connected layers in AlexNet and ResNet for the cases before and




(a) AlexNet




(b) ResNet-18

Figure 11: Density distribution of convolution and fully-connected layers in (a) AlexNet and (b) ResNet-18 structure. The weight density of the pre-trained model is when the network just loads the parameters from the pre-trained model without fine-tuning. Ternarized model: fl=FP, ll=FP refers to the density distribution of the ternarized model with the first and last layer in full precision. Ternarized model: fl=Tern., ll=Tern. denotes that the density distribution of the ternarized model with first and last layer in ternary representation.

after the model ternarization. As shown in both Section 2.5.2 and Section 2.5.2, the weight density of the first and last layer is raised to a higher level after ternarization in comparison to their initial density. Here the density on the pre-trained model refers to the neural network that initialized from the full-precision pre-trained model, then we directly apply our ternarization method as introduced in Section 2.3 without fine-tuning the weights. We calculate the average density of all the convolution and fully-connected layers, which are reported in Table 10. It shows that after ternarization, the density of the first and last layer is much higher than the average density and other layers. A potential method to further compress the network model is to use density as an indicator to choose different expansion factors for each layer.

### 2.5.3 REL Versus Multi-bit Weight

Table 11: Inference accuracy of ResNet-18 on ImageNet with multi-bit [39] weight or REL.

| | Multi-bit [39] | REL (this work) | |
|---|---|---|---|
| Config. | 3-bit | $t_{ex}$=1 | $t_{ex}$=2 |
| Accuracy | 65.70/86.82 | 66.01/86.78 | 68.05/88.04 |
| Comp. rate | 10.7× | 16× | 8× |

In comparison to directly using multiple bits (multi-bit) weight, using REL is advantageous in terms of both accuracy and hardware implementations. First, REL shows better performance over the multi-bit method adopted from Dorefa-Net [39] on ImageNet. Second, from the hardware perspective, REL preserves all network weights in the ternary format (i.e. -1, 0, +1), enabling the implementation of convolutions through sparse addition operations without multiplication. Increasing the number of

expansion layers ($t_{ex}$) merely requires more convolution kernels when one deploys the network in FPGA/ASIC. It is noteworthy that the REL with a larger threshold factor owns higher sparsity, which requires less computational resources. On the contrary, when more bits are added in multi-bit, the convolution computation goes back to multiplication and addition operations, consuming more energy and chip area. Finally, both our work (Section 2.5.2) and the deep compression [20] show that a deep neural network has different redundancies from layer to layer. If one employs the multi-bit scheme, we have to re-design the computing hardware with different bit-widths for the layers. Beyond that, since zero-value weights in REL with a smaller threshold will be staying in zero in REL with a larger threshold (as shown in Section 2.4.2.3 and Section 2.4.2.3), the model compression rate can be further improved with proper weight sparse encoding method.

### 2.5.4 Hardware Resource Utilization Efficiency

Table 12: FPGA resource utilization (Kintex-7 XC7K480T) for AlexNet last layer

| Weight type | MACs | | LUT (slices) | | DSP (slices) | |
|---|---|---|---|---|---|---|
| | Multiplier | Adder | required | available | required | available |
| FP | 100 | 100 | 49600 | 74650 | 200 | 1920 |
| Tern. | 0 | 90 | 23490 | 74650 | 0 | 1920 |

As we discussed in section 2.4.2.1, fully ternarizing the first and last layer is greatly beneficial for developing the corresponding hardware accelerator on FPGA or ASIC, owing to the elimination of floating-point convolution computing core. Here in this work, we analyze the deployment of the last fully connected layer of AlexNet into a XILINX high-end FPGA (Kintex-7 XC7K480T) as an example. As reported by [91],

floating-point adder cost 261 LUT slices on Kintex-7, while the floating-point multiplier takes 235 LUT slices with additional 2 DSP48Es. The weights of last fully-connected layer in AlexNet is in the dimension of $4096 \times 1000$, which requires 1000 *MACs* [92] if performing the computation in the extreme parallel manner. However, due to the FPGA resource limitation, we consider the design scheme that allows 100 MACs for floating-point weight convolution, while the rest resources can only accommodate 90 MACs for ternary weight convolution. For the neural network with both ternary and full precision weights, balancing the hardware resources to build MAC cores for ternary and full precision weight becomes extremely difficult. Increasing the number of ternary weight MAC cores does improve hardware resource utilization. However, it has to reduce the allocated floating-point MACs, which at the same time reduces the computation parallelism and thus increases whole system latency. On the contrary, if we keep the current hardware resource allocation plan, the hardware utilization efficiency will be extremely low especially when the networks are going deeper. Then, it can be seen that fully ternarizing the whole network is extremely important for algorithm deployment in a resource-limited embedded system although the theoretical model compression rate does not increase too much.

## 2.6 Summary

In this chapter, we first briefly go through the basics of neural network quantization. Then, we have explicitly optimized the current DNN ternarization scheme with techniques like statistical weight scaling and REL. On the image classification task, a series of comprehensive experiments are conducted to show that our method can achieve state-of-the-art accuracy on both small dataset like CIFAR-10, and large dataset like

ImageNet. Beyond that, we examined the accuracy with/without the first&last layer ternarization. Owing to that the statistical weight scaling is used through the entire network, our ternarized network does not encounter serious accuracy degradation even with ternarized weight for the first&last layer.

Chapter 3

# EFFICIENT NEURAL INFERENCE VIA ANALOG IN-MEMORY COMPUTING

In the prior chapter, we have covered the basics of neural network compression leveraging the weight and activation quantization. Neural networks severing on the general-purpose computing hardware (e.g., Graphics Processing Unit, GPU) could significantly benefit from the performance boost resulting from the model quantization. In order to further enhance the inference efficiency, accelerating the neural network inference with the Resistive Random-Access-Memory (ReRAM) crossbar is a promising direction [42], [43], [93], since computations are performed in memory without expensive long-distance data communication. Nevertheless, various non-ideal effects of the ReRAM crossbar are hampering the inference accuracy of the deployed neural network in the practical scenario. In this chapter, we will focus on developing effective countermeasures to mitigate the ReRAM crossbar non-ideal effects.

## 3.1 Preliminaries

Resistive crossbar memory, as one the most popular memory array structure, has drawn great research interest owing to its high memory accessing bandwidth and *in-situ* computing ability [43], [94]. More importantly, its current-mode weighted summation operation intrinsically matches the dominant Multiplication-and-Accumulation (MAC) in an artificial neural network, making it one of the most promising candidates as the basic computing unit for neural network accelerator design. Various

crossbar-based neural network accelerator designs with different technologies, such as SRAM [95], DRAM [96], ReRAM [97] and MRAM [77], have been proposed recently, which show significant performance improvement for neural network inference. Resistive Random-Access-Memory (ReRAM) is a two-terminal device with programmable resistance, which is taken as the target device in this work. However, many non-ideal effects, such as wire resistance, Stuck-At-Fault (SAF), thermal noise, shot noise, random telegraph noise, and etc. [65], are hampering the progress of real hardware implementation of large-scale deep neural network (DNN) on ReRAM crossbar-based accelerator. Many recent works [62]–[65] have investigated such issues and proposed various solutions from either hardware or software perspectives, which are summarized as follows.

**Hardware Approach.** In [98], Xu et al. investigate the impact of different resistance allocation schemes, programming strategies, peripheral designs, and material selections on the area, latency, power, and reliability of ReRAM crossbar. The IR-drop caused by the wire resistance of the crossbar is modeled by Liu et al. in [58].

**Software Approach.** Other recent works [62]–[65] have adopted different methods that all require retraining the target neural network's weight to be mapped in a crossbar array w.r.t various non-ideal effects for a specific device. However, the main drawback of such a method is that compute-intensive additional network retraining is required for each specific crossbar accelerator, which is definitely not cost-efficient and not practical for future deep neural network deployment into different crossbar accelerators. Moreover, typical network retraining to get reconstructed weights only considers and adapts the deterministic non-ideal effect (e.g., SAF defects). While, incorporation of many other non-ideal effects (e.g., thermal noise) with stochastic behavior or IR-drop still remains unsolved. Therefore, rather than network retraining to

get reconstructed weights every time for a new crossbar accelerator as discussed above, we propose different techniques for each type of non-ideal effect, which only needs one-time optimization for target DNN.

## 3.2 Non-Ideal Effects and Modeling of ReRAM Crossbar

Generally, the non-ideal effects of ReRAM crossbar can be divided into two categories and modeled as deterministic noise $n_{\mathrm{d}}$ and stochastic noise $n_{\mathrm{s}}$, respectively. In this work, we consider Stuck-At-Fault (SAF) as the deterministic non-ideal effect, while the stochastic non-ideal effects include effects of crossbar wire resistance (i.e., IR-drop), thermal noise, shot noise and Random Telegraph Noise (RTN). All the aforementioned non-ideal effects are explicitly discussed in the following subsections.

### 3.2.1 Deterministic Noise Modeling

#### 3.2.1.1 Device Defects

There are two kinds of Stuck-At-Fault (SAF) defects, which are Stuck-At-zero (SA0) and Stuck-At-one (SA1) respectively [56], [57]. The SA0 defect is normally caused by short defects, which makes parts of the ReRAM cells always at high conductance state (i.e., $G_{\mathrm{max}}$). The SA1 defect is resulting from the cell permanent damage and broken selector or wire, which makes parts of the ReRAM cells stuck at low conductance state (i.e., $G_{\mathrm{min}}$). The ReRAM fabrication results in [99] show 1.75% and 9.04% defect rate for SA0 and SA1 respectively. The SA0/SA1 defect rates around the 1.75%/9.04% are taken as the region of interest in this work.

### 3.2.2 Stochastic Noise Modeling

#### 3.2.2.1 Wire Resistance

In this work, the reason we consider the wire resistance as a stochastic noise term is that the crossbar output current degradation is highly correlated with both input voltage and ReRAM conductance distribution. Modeling it as a deterministic term is extremely computing expensive and not feasible for large scale DNN mapping. For example, in [58], Liu et al. employ a gradient search method to compensate for the weight matrix for IR-drop. However, the process needs to be repeated whenever the inputs of crossbar changes, which require massive computing resources and training time. While in this work, we propose to leverage the statistics of IR drop caused current drift of each crossbar from PytorX as a stochastic noise source in the training of DNN. Such an approximate method is fast and could eventually regularize the trained network to adapt to the IR drop of the crossbar. The details of our proposed IR drop modeling methods are shown later in Section 3.4.3.

#### 3.2.2.2 Thermal Noise and Shot Noise

Thermal noise is normally caused by thermal agitation of the electrical carriers, while the shot noise is induced by the random arrival of the carriers [59]. Both of them can be modeled with random current source following zero mean Gaussian distribution. Thus we merge them together as a single current source presented in parallel with ReRAM and the Root-Mean-Square (RMS) current, which can be expressed as:

$$i_{\text{rms}} = \sqrt{Gf(4K_{\text{B}}T + 2qV)} \tag{3.1}$$

where $G$ is the ReRAM conductance, $f$ is the operating frequency of the crossbar, $K_{\mathrm{B}}$ is Boltzmann constant, $T$ is the temperature in Kelvin, $q$ is electron charge, $V$ is the voltage drop across the ReRAM two terminals. Furthermore, we also include the programming variation of ReRAM in our noise model. We consider the programming variation follows Gaussian distribution $\mathcal{N}(0, \sigma^2)$, where $\sigma = \Delta G/3$ and $\Delta G$ is the ReRAM conductance resolution. Then, the equivalent standard deviation of ReRAM conductance can be described as:

$$g_{\text{rms}} = \frac{i_{\text{rms}}}{V} = \sqrt{\frac{Gf(4K_{\mathrm{B}}T + 2qV)}{V^2} + \left(\frac{\Delta G}{3}\right)^2} \tag{3.2}$$

#### 3.2.2.3 Random Telegraph Noise

Random Telegraph Noise (RTN) constitutes a major cause of the errors in non-volatile resistive devices by temporarily and randomly reducing the resistance of the ReRAMs [61]. In this work, we adopt the RTN model proposed by Ielmini et al. in [60], where the average ReRAM resistance change ($R_{\text{rtn}}$) strongly depends on the current resistance state of ReRAM ($R$). The relation can be described as $R_{\text{rtn}}/R = aR + b$, where $a$ and $b$ are fitting parameters. Converting the resistance into conductance, we can obtain $G_{\text{rtn}}/G = \frac{bG+a}{G-(bG+a)}$. We extract $a = 1.662e-7$ and $b = 0.0015$ from the data reported by [60], then model the RTN as a Poisson process [61]. Thus, the actual ReRAM conductance value under RTN ($G_{\text{act}}$) can be mathematically written as:

$$G_{\text{act}} = \begin{cases} G + G_{\text{rtn}} & \tau < 0.5 \\ G & otherwise \end{cases} \tag{3.3}$$

where $\tau$ is a random number that uniformly distributed between (0,1) (data from [61]).

## 3.3 Crossbar based NN-accelerator

In this section, we will first introduce the single ReRAM crossbar design as a dot-product engine. Then, the network partition method we adopted for mapping the large scale network on multiple arrays is introduced in the following subsection. Note that, we consider there exists a digital arithmetic unit as the co-processor within the ReRAM crossbar-based accelerator, for signal scaling, addition with bias and other computations like Batch-Normalization layers.

### 3.3.1 Single Array as Dot-product Engine

The primary computation of convolution and fully connected layer in deep convolution neural network is the weighted summation with bias offset, which can be rewritten as the element-wise dot-product format:

$$y = \boldsymbol{w}^T \cdot \boldsymbol{x} + b = \begin{bmatrix} \boldsymbol{w}^T \\ 1 \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{x} & b \end{bmatrix} \tag{3.4}$$

where $\boldsymbol{w}$ and $\boldsymbol{x}$ are vectorized weights and inputs respectively, and $b$ is the bias term. For hardware mapping purpose, both weight ($w \in \mathbb{Z} \cdot \Delta x$) and input ($x \in \mathbb{Z} \cdot \Delta w$) are quantized into fixed-point representation, where $\Delta x$ and $\Delta w$ are the fixed-point quantization step size. In this work, we choose 8-bit for both the inputs and the weights, and we emit the bias term where the computation is performed by the digital co-processor. Thus, the above equation can be rewritten as:

$$y = (\Delta w \cdot \Delta x) \sum_i \hat{w}_i \cdot \hat{x}_i \tag{3.5}$$

The ReRAM crossbar is taken as the analog dot-product engine to accelerate the computation of Eq. (3.5) as shown in Fig. 12. Owing to the weights can be either positive

or negative, each weight is represented by two ReRAM cells (i.e., $G^+_{i,j}$ and $G^-_{i,j}$) allocated in positive and negative array. As shown in Fig. 12, the input $x_i$ is encoded as binary bit-strings $\text{in}_i[n]$ for crossbar input, where each one has 8-bit digits ($n = 8$) in 2's complement format. Such binary encoded inputs are converted by the DACs, where the output voltage of the DAC on $i$-th row can be expressed as:

$$V_{\text{DAC},i} = V_{\text{ref}} + \Delta V_{\text{DAC}} \cdot \hat{x}_i \tag{3.6}$$

where $V_{\text{ref}} = V_{\text{DD}}/2$ is the reference voltage, and $\Delta V_{\text{DAC}}$ is the minimum voltage stage of the DAC. Therefore, the current forward into the differential ADC for $j$-th column pair (i.e., two corresponding columns in positive and negative array) can be described as:

$$I_{\text{ADC},j} = \sum_{i=1}^{M} \left( (V_{\text{DAC},i} - V_{\text{ref}}) \cdot (G^+_{i,j} - G^-_{i,j}) \right) \tag{3.7}$$

For ReRAM crossbar programming, there exist two mapping schemes which are equal-$\Delta R$ and equal-$\Delta G$ respective. Hereby we adopt the more straight-forward equal-$\Delta G$ mapping scheme, since the equal-$\Delta R$ introduces an undesired device-oriented non-uniform quantization. In order to properly mapping the Eq. (3.5) onto the Eq. (3.7), we rewrite the function as follows:

$$G^+_{i,j} - G^-_{i,j} = \Delta G \cdot \hat{w} \tag{3.8}$$

$$G^+, G^- = \begin{cases} \Delta G \cdot \hat{w} + G_{\text{min}}, G_{\text{min}} & \text{if } \hat{w} \geq 0 \\ G_{\text{min}}, \Delta G \cdot |\hat{w}| + G_{\text{min}} & \text{if } \hat{w} < 0 \end{cases} \tag{3.9}$$

where $\Delta G$ is the conductance step size of the programmable ReRAM cell. Note that here the conductance $G^+/G^-$ is the series combination of the on-conductance of the selector $G_{\text{on}}$ and the ReRAM programmed value $G_{i,j}$. Thus, Eq. (3.7) can be reformatted

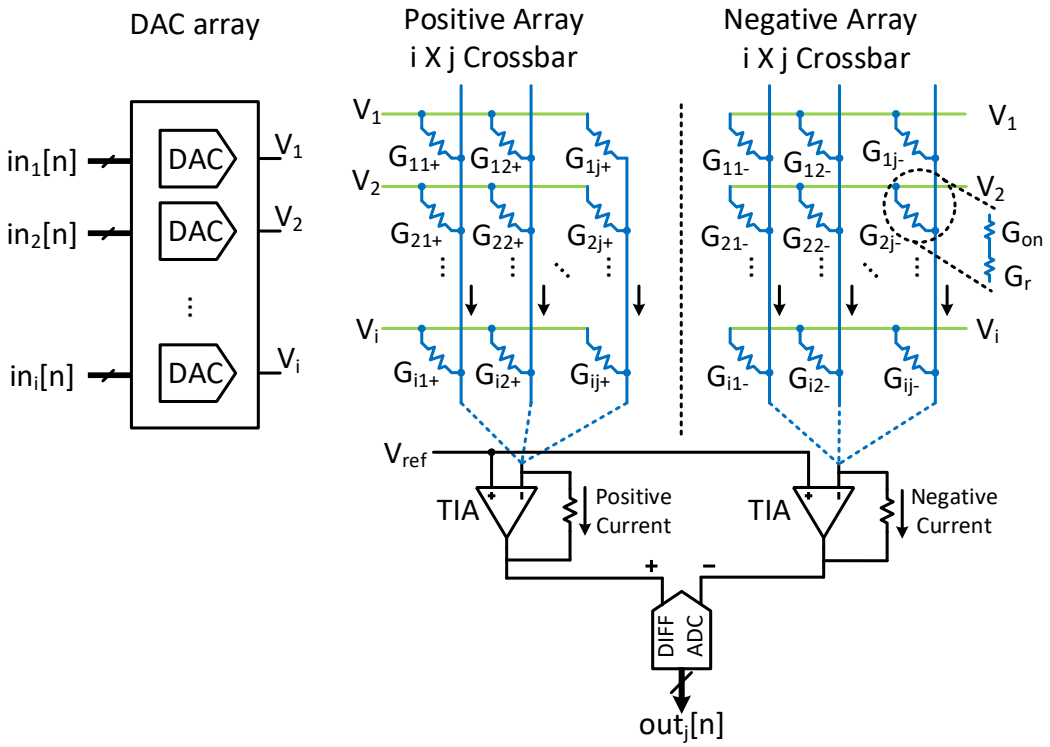



Figure 12: Hardware implementation of single $M \times M$ ReRAM crossbar array pair (positive and negative array) as analog dot-product engine. ReRAM selector is modeled as $G_{\text{on}}$ cascaded with ReRAM conductance $G_{\text{r}}$.

as:

$$I_{\text{ADC},j} = (\Delta V_{\text{DAC}} \cdot \Delta G) \sum_{i=1}^{M} \hat{x} \cdot \hat{w} \tag{3.10}$$

Then, with the ADC at the output-end for performing current sensing and quantization function[8], where the LSB current of ADC is configured as $\Delta I_{\text{ADC}} = k \cdot \Delta V_{\text{DAC}} \cdot \Delta G$ , the final output is:

$$\hat{y} = \text{round}(\frac{I_{\text{ADC},j}}{\Delta I_{\text{ADC}}}) = \text{round}(\frac{1}{k} \sum_{i=1}^{M} \hat{x} \cdot \hat{w}) \tag{3.11}$$

In this work, we consider $\Delta V_{\text{DAC}}$ and $\Delta G$ are fixed by hardware, while $\Delta I_{\text{ADC}}$ is configurable through tuning the reference current source of ADC. To prevent the ac-

[8]In this work, we use the Eq. (3.11) for simplicity. For a more commonly used ADC quantization, $\hat{y} = \text{round}(I_{ADC,j}/\Delta I_{\text{ADC}} - 0.5) + 0.5$ is also an option.

**Algorithm 1** DNN quantization training scheme.

**Require:** Layer-wise quantization step of input $\Delta x$, weight $\Delta w$ and ADC $\Delta I_{\text{ADC}}$. Resolution of input, weight, ADC are in $N_{\text{B}}^{\text{dac}}$, $N_{\text{B}}^{\text{rram}}$ and $N_{\text{B}}^{\text{adc}}$ bits. Number of training batches $N_{\text{train}}$.

**Ensure:** Minimize the loss and obtain fixed $\Delta x$, $\Delta w$ and $\Delta I_{\text{ADC}}$.

{1. Training phase of $l$-th layer in epoch $t$}

Initialization: $S_{\text{x}} \leftarrow 0$; $S_{\text{I}} \leftarrow S_{\text{I}} + \Delta I_{\text{ADC}}$

**for** $i$ := 1 to $N_{\text{train}}$ **do**

  Update $\boldsymbol{w}_l$ through back-propagation

  $\Delta w_l \leftarrow \max(|\boldsymbol{w}_l|)/2^{N_{\text{B}}^{\text{rram}}}$

  $\Delta x_l \leftarrow \max(|\boldsymbol{x}_l|)/2^{(N_{\text{B}}^{\text{dac}}-1)}$

  Compute $\boldsymbol{I}_{\text{ADC}}$ ▷ Eq. (3.10)

  $\Delta I_{\text{ADC}} \leftarrow \max(|\boldsymbol{I}_{\text{ADC}}|)/2^{(N_{\text{B}}^{\text{adc}}-1)}$

  $S_{\text{x}} \leftarrow S_{\text{x}} + \Delta x_l$; $S_{\text{I}} \leftarrow S_{\text{I}} + \Delta I_{\text{ADC}}$

**end for**

{2. Test phase}

$\Delta x_l \leftarrow S_x/N_{\text{train}}$; $\Delta I_{\text{ADC}} \leftarrow S_{\text{I}}/N_{\text{train}}$

curacy degradation caused by the quantization error, we also propose a crossbar-aware network quantization method as described in Algorithm 1. Rather than minimizing the quantization error w.r.t each input, our method optimizes the layer-wise $\Delta w$, $\Delta x$, and $k$, then return them as a fixed value. Thus, the computation workload on digital co-processor is minimized.

### 3.3.2 Network Partition on Matrix Arrays

Nowadays, the parametric layers of the state-of-the-art DNN normally contain a large number of weights which exceeds the size of a single crossbar. Thus, each DNN layer may require multiple crossbar arrays for weight accommodation. A network partition technique is necessary here for efficient weight mapping and network computation. For simplicity, we adopt the naive network partition method similar as introduced in [100], which converts the weight tensor of each convolution/fully-connected layer into

two 4-D Matrix Arrays with shape $\{N_{row},N_{col},C_{size},C_{size}\}$ as $G^+$ and $G^-$ respectively, where $N_{row} \times N_{col}$ is the number of crossbar arrays used for one layer.

## 3.4 End-to-End Network Adaption

In this section, we will explicitly discuss the impacts of different non-ideal effects on ReRAM crossbar based Deep Neural Network acceleration, and the effectiveness of our proposed countermeasures for preventing accuracy degradation.

### 3.4.1 Simulation Framework and Configurations

We divide the experimental setup into hardware and software parts for the clarification of the tool-chain and models used in this work.

**Hardware:** The state-of-the-art fabrication result has shown the 6-bit [101] to 7-bit [102] ReRAM programmable resistance level. In this work, we choose a moderate resolutions for ADC, DAC and ReRAM (i.e., 8-8-7 bit), other chosen parameters are tabulated in Table 13. Partial of the parameters chosen for the non-ideal effects setup are provided in Section 3.2. The ReRAM crossbar-based neural network accelerator is operating as introduced in Section 3.3.

**Software:** To perform a comprehensive investigation on the ReRAM crossbar based neural network accelerator under various non-ideal effects, we build a comprehensive simulation framework called *PytorX*, which is implemented in Python and based on PyTorch, a mainstream DNN training and evaluation platform. It models the crossbar computation based on the discussion and equations in Section 3.3, including signal/unit conversion (i.e., DAC, ADC, and weight mapping), weight partition on mul-

Table 13: Parameters of ReRAM crossbar system.

| Symbol | Description | Value |
|---|---|---|
| $f$ | Operating frequency | 0.01∼1 GHz |
| $V_{\text{dd}}/V_{\text{ref}}$ | Supply/reference voltage | 3.3/1.67 V |
| $G_{\text{min}}/G_{\text{max}}$ | ReRAM min/max conductance | 333/0.33 $\mu$S |
| $\Delta G$ | Conductance step size | 2.601 $\mu$S |
| $N_{\text{B}}^{\{\text{adc,dac}\}}$ | Resolution of ADC/DAC | 8/8 bit |
| $T$ | Temperature | 300∼350 K |
| $C_{\text{size}}$ | Crossbar Dimension | 32, 64, 128 |

tiple arrays, etc. PytorX is a GPU-favored framework and fully relies on the tensor operation, thanks to the rich APIs provided by PyTorch. To demonstrate the effectiveness of our proposed method, we take the classical object recognition task as an example. A variety of network structures (LeNet-5 variant [9] and ResNet-20 [2]) on different scales datasets (MNIST [103] and CIFAR-10 [104]) are studied in the following sections. Hyper-parameters configuration and image argumentation methods adopted in this work are identical to what used in the network's original paper, which will not be explicitly listed here.

### 3.4.2 Error Correction for SAF

Many previous studies [57], [64] only discuss the effect of SAF on network inference using a simple 2-layers fully-connected network, where the results show that SAF indeed affects the accuracy but with relatively low degradation. However, what we found in our experiments is that such accuracy degradation drastically increases with a wider and deeper network structure on a larger dataset. Fig. 13 depicts the test ac-

[9] https://github.com/pytorch/examples/blob/master/mnist/main.py

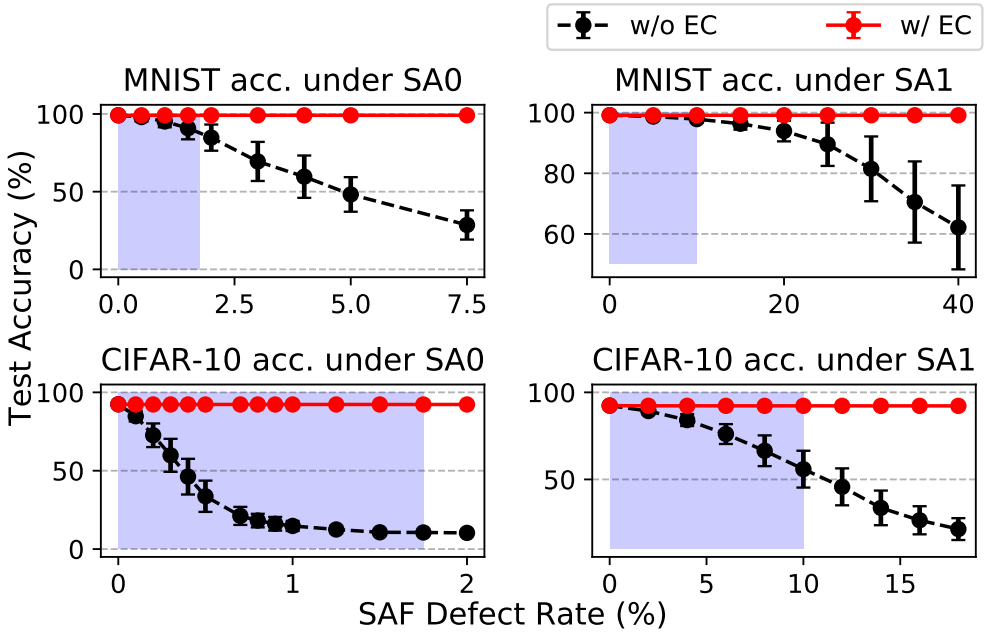



Figure 13: Test accuracy versus SA0/SA1 rate, w/o or w/ Error Correction (EC). (Top) LeNet-5 on MNIST dataset; (Bottom) ResNet-20 on CIFAR-10 dataset. Error-bar denotes mean±std with 100 trials, and crossbar size is 64. Regions with blue shadow are regions of interest.

curacy on MNIST and CIFAR-10 dataset using LeNet-5 and ResNet-20 respectively, where the blue shadows indicate accuracy degradation caused by SAF in the deep neural network. Note that, the SAF rate we used in is 1.75% for SA0 and 9% for SA1, respectively. As shown in Fig. 13, when directly mapping the network into a crossbar system without any error correction technique, only 1% SA0 defect rate can lead to a complete system malfunction on CIFAR-10 dataset with ResNet-20.

As the countermeasure to both SA0 and SA1, in this work, we propose a digital SAF Error Correction (EC) solution to compensate the computation error in the digital-end. Such method can be generally described into steps: **1)** Profiling the current SAF status of ReRAM crossbar system, then indexing the ReRAM cells with SA0/SA1 defects using binary tensor $\boldsymbol{i}^{\text{sa0}}/\boldsymbol{i}^{\text{sa1}}$; **2)** Calculating the crossbar output difference $\boldsymbol{y}_{\text{ec}}$ caused

by SAF w.r.t the outputs in the ideal case, where the detail is elaborated in Algorithm 2; **3)** Adding the yield $\boldsymbol{y}_{\text{ec}}$ at the crossbar output utilizing the digital units. Normally, the addition operation in last step can integrate with the following Batch-Norm layer (i.e., Affine function). For CIFAR-10 test accuracy reported in Fig. 13, the SAF-free accuracy is 92.39%. With our error correction method, the worst-case accuracy we obtain is $92.23 \pm 0.08\%$, where the accuracy gap is negligible. Moreover, the binary SAF indexing tensor can use the sparse encoding for efficient storage and computation. Note that, only one-time profiling is needed for each crossbar, instead of compute-intensive whole DNN re-training as discussed in previous works.

**Algorithm 2** Error Correction (EC) for Stuck-At-Fault (SA0/SA1).

**Require:** input voltage tensor $\boldsymbol{V}$, ideal weight conductance tensor w/o SAF $\boldsymbol{G}_+$ and $\boldsymbol{G}_-$, non-ideal weight conductance tensor w/ SAF $\boldsymbol{G}_+^{\text{saf}}$ and $\boldsymbol{G}_-^{\text{saf}}$. SA0 indexing tensor $\boldsymbol{i}_+^{\text{sa0}}$ and $\boldsymbol{i}_-^{\text{sa0}}$, SA1 indexing tenor $\boldsymbol{i}_+^{\text{sa1}}$ and $\boldsymbol{i}_-^{\text{sa1}}$. SA0 and SA1 leads to $G_{\text{max}}$ and $G_{\text{min}}$ respectively. ADC current-to-digital conversion $f_{\text{adc}}(\cdot)$.

**Ensure:** for the given input $\boldsymbol{V}$, the output of ReRAM crossbar is corrected using error compensation w.r.t the profiled SA0 and SA1 information.

$\boldsymbol{y} \leftarrow f_{\text{adc}}(V \times \boldsymbol{G}_+^{\text{saf}} - V \times \boldsymbol{G}_-^{\text{saf}})$ ▷ Crossbar output

$\boldsymbol{G}_+^{\text{diff}} \leftarrow (\boldsymbol{G}_+ - G_{\text{max}}) \cdot \boldsymbol{i}_+^{\text{sa0}} + (\boldsymbol{G}_+ - \boldsymbol{G}_{\text{min}}) \cdot \boldsymbol{i}_+^{\text{sa1}}$

$\boldsymbol{G}_-^{\text{diff}} \leftarrow (\boldsymbol{G}_- - G_{\text{max}}) \cdot \boldsymbol{i}_-^{\text{sa0}} + (\boldsymbol{G}_- - \boldsymbol{G}_{\text{min}}) \cdot \boldsymbol{i}_-^{\text{sa1}}$

$\boldsymbol{y}_{\text{ec}} \leftarrow f_{\text{adc}}(V \times \boldsymbol{G}_+^{\text{diff}} - V \times \boldsymbol{G}_-^{\text{diff}})$ ▷ output diff. w.r.t SAF

**return** $\boldsymbol{y}_{\text{out}} \leftarrow \boldsymbol{y} + \boldsymbol{y}_{\text{ec}}$ ▷ Digital error compensation

### 3.4.3 Noise Injection Adaption for IR-drop

The IR-drop caused by wire resistance is another dominant factor which may cause the system malfunction. With a specific wire technology, the IR-drop is not only determined by the crossbar dimension, but strongly correlated to both input voltage magnitudes and conductance distribution of ReRAM cells. To visualize the impact of the

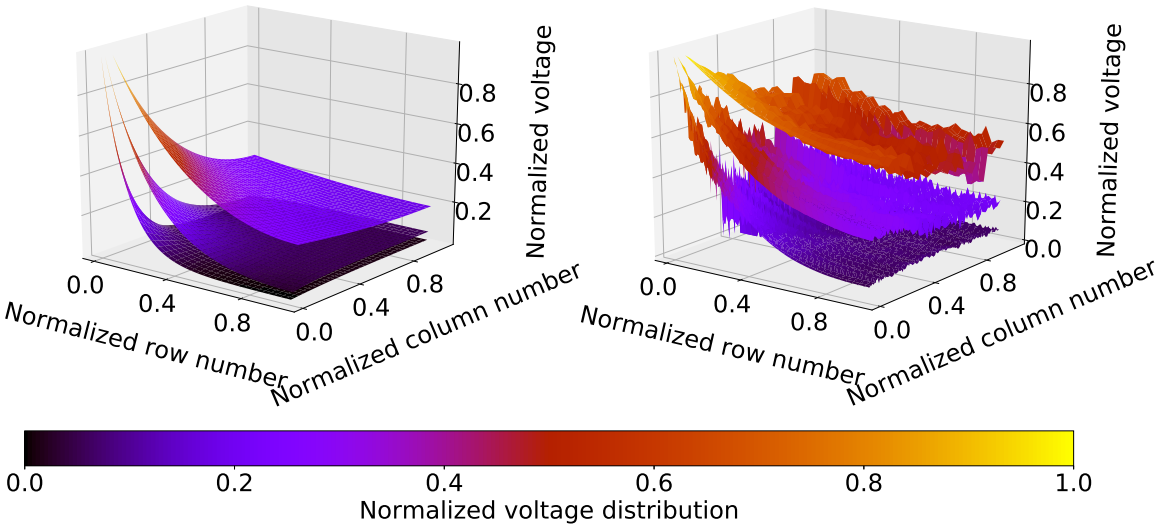



Figure 14: Distribution of Voltage drop of crossbar with different dimensions. Surfaces from top down are $32 \times 32$, $64 \times 64$ and $128 \times 128$ size respectively. (Left) is worst case, (Right) is normal case

crossbar dimension, we plot the distribution of normalized voltage drop ($V'/V$) across ReRAM cells in Fig. 14, where $V'$ is the voltage drop considering IR-drop, while $V$ is the ideal case without IR-drop. Fig. 14 indicates that choosing $128 \times 128$ as crossbar size is impractical for an accelerator design due to a huge IR drop.

To study the impact of IR-drop on DNN inference accuracy, we integrate a dynamic crossbar solver into our PytorX simulation framework, based on the simplified Modified Nodal Analysis [105]. The test accuracy on MNIST and CIFAR-10 with IR-drop considered is reported in Table 14, where directly mapping the network on $64 \times 64$ crossbar results in more than 65% accuracy degradation. For mitigating such significant accuracy loss, we propose to optimize DNN training considering such IR-drop effects, by compensating real-time current shift for each crossbar to tune DNN weights and make it adaptive to existing IR drop effect. However, the biggest problem is that the existing crossbar IR drop solver requires massive computation resources to get real-time shifted current based on current inputs, which also grows exponentially

with larger crossbar size. In our experiment, a simple LeNet5 training takes all of the 64GB main memory and costs days to converge in a 4 NVIDIA TitanX GPU workstation, which is too computing-intensive if considering such accurate IR-drop effects. Note that, it will become even worse if the network becomes deeper and wider.

Thus, we propose a Noise Injection Adaption (NIA) method which can statistically approximate the effect of IR-drop and incorporate it into network training, for balancing computation resources and the accuracy of the trained model. Taking LetNet5 on MNIST as an example, our NIA can be divided into three steps: (1) we first train a LetNet5 without considering IR-drop during training. Then, with the trained LetNet5 mapped on the crossbar matrix arrays in our PytorX, we can collect the crossbar output current shift with or without IR-drop for all of the testing data; (2) we approximate the effect of IR-drop as a Gaussian noise source at each crossbar output end, with crossbar-wise mean and standard deviation extracted from the collected statistics; (3) we train the network again with such approximated additive IR drop noise applied at each output of crossbar arrays. The only overhead of our NIA method is a network optimization step which has similar timing cost as a typical DNN training. After the network optimization with NIA, accuracy degradation is significantly reduced as reported in Table 14.

### 3.4.4 System Configuration for Other Stochastic Non-ideal Effects

As discussed in Section 3.2.2, we categorize the thermal noise, shot noise and random telegraph noise as ReRAM stochastic noise to study in this work, where the actual

Table 14: Test accuracy on MNIST with various crossbar dimensions and our proposed methods

| Dataset (crossbar size) | MINIST (32×32) | MNIST (64×64) |
|---|---|---|
| Software | 99.04% | 99.1% |
| Direct Mapping with EC | 96.2% | **32%** |
| Optimized with NIA | 99.1% | **98.3%** |
| Optimized with NIA + EC + frequency tuning | 99.0% | 98.1% |

ReRAM conductance with additive ensemble stochastic noise can be modeled as:

$$G_{\text{act}} = G + G_{\text{g}} + G_{\text{p}} \tag{3.12}$$

where $G_{\text{g}} \sim \mathcal{N}(0, g_{\text{rms}}^2)$ is the Gaussian term as in Eq. (3.2), and Poisson term is $G_{\text{p}} = G_{\text{rtn}}$ under the Poisson process as discussed in Eq. (3.3). It is noteworthy that the Gaussian term is highly correlated with both the ReRAM programmed conductance and crossbar input voltages, which makes the conductance variation of ReRAM cells differs w.r.t each input, which is considered in the simulation. We first try to apply the NIA technique to retrain the neural network for adapting those stochastic terms described in Eq. (3.12). However, our simulation turns out injecting a Gaussian noise $\mathcal{N}(0, \Delta G)$ to the weight equivalent conductance $G$ does not reduce the accuracy degradation. On the contrary, it reduces the inference accuracy. Our explanation to such observation is that injecting Gaussian noise on the weight performs L-2 norm regularization, which leads to entire ReRAM arrays have smaller conductance. However, according to RTN modeling discussed in Section 3.2.2.3, ReRAM cell with smaller $G$ owns larger RTN variation which leads to incorrect inference results.

Therefore, in this subsection, rather than trying to adapt the neural network to the stochastic noise terms in Eq. (3.12), we attempt to mitigate thermal noise, shot noise

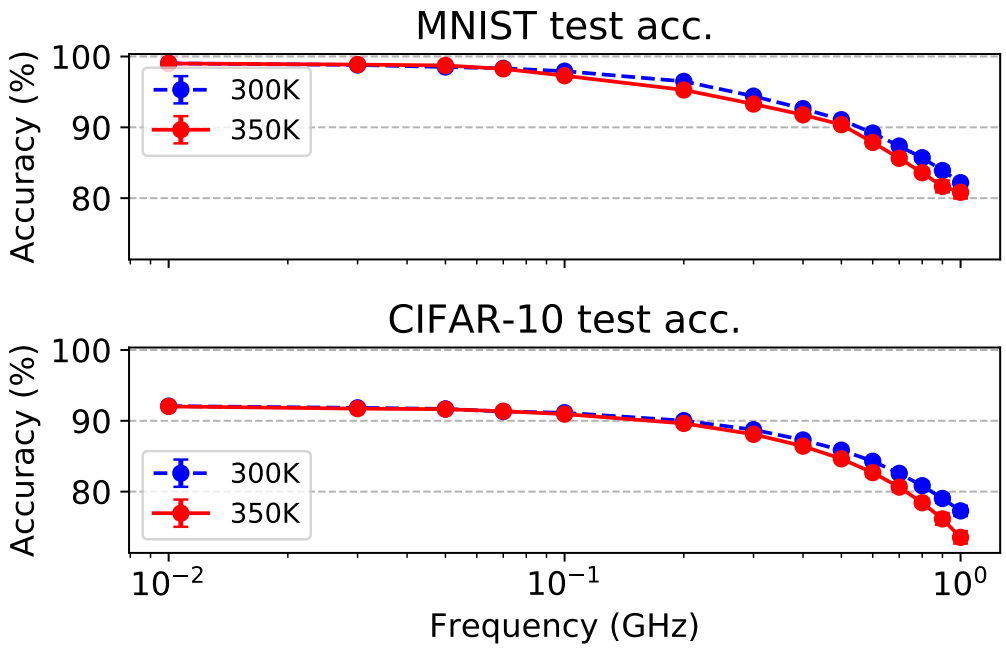



Figure 15: Test accuracy on MNIST and CIFAR-10 dataset versus operating frequency $f$.

and RTN side effects simultaneously through optimizing system operating frequency. The examination of the inference accuracy of the crossbar system under varying operating frequency ($f$) and temperature ($T$) is shown in Section 3.4.4. It shows that lowering the operating frequency can effectively reduce the thermal noise and shot noise caused accuracy degradation. Then, the left-over terms of programming variation and RTN do not show significant impacts on the inference accuracy when $f$ is reduced. We also perform a Monte Carlo simulation with 10000 trails to observe ReRAM conductance variation when $G = G_{\text{min}}$. As shown in Section 3.5, when the $f$ is lowered to 10MHz, the conductance variation is almost located within the quantization boundary $(-\Delta G, +\Delta G)$.

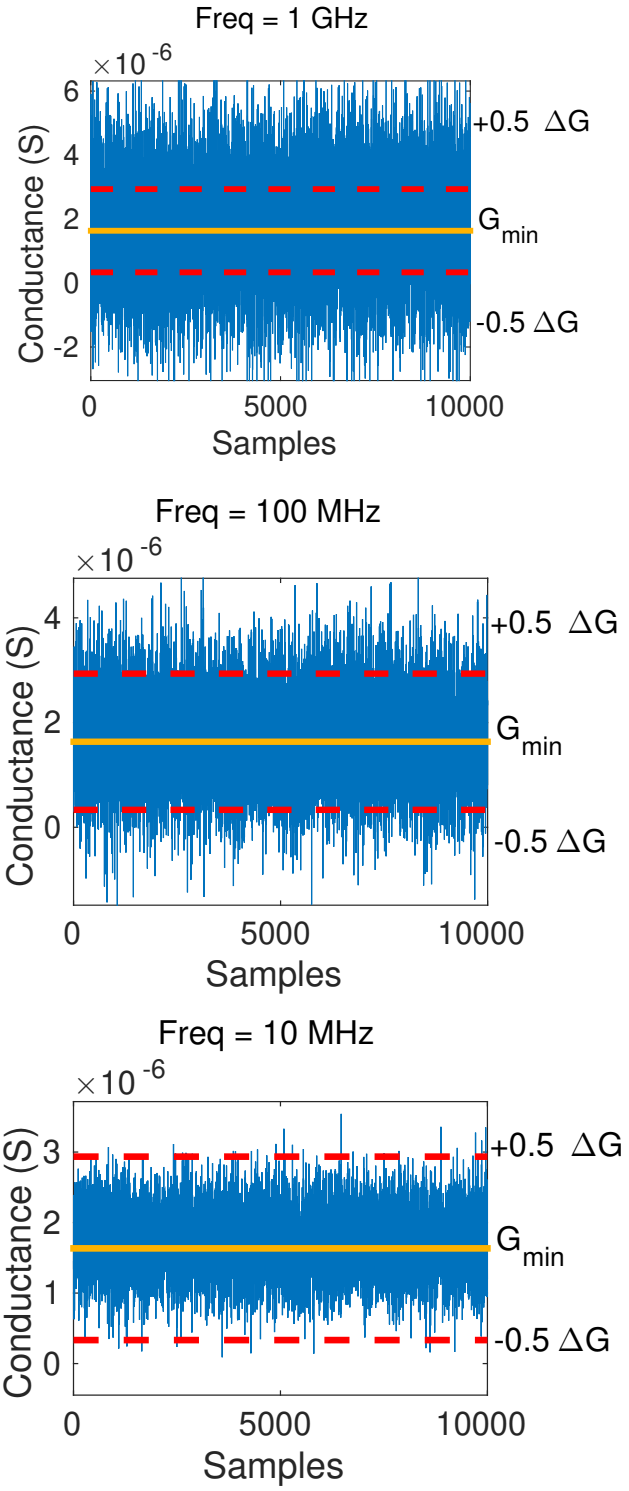



Figure 16: Monte Carlo simulation of the $G_{\text{act}}$ under $f = 1GHz, 100MHz, 10MHz$, with 10000 trials. The solid line indicate the ideal conductance ($G_{\text{min}}$ in this simulation), and dashed line is the quantization boundary.

## 3.5 Summary

Through PytorX, we can perform a comprehensive and accurate simulation of large-scale DNN that mapped to the ReRAM crossbar-based accelerator. To overcome SAF effects, we proposed a digital SAF error correction algorithm to achieve almost no accuracy degradation in DNN mapping without the need for network retraining. To overcome the IR drop effect, we proposed a noise injection adaption method to model IR drop as a stochastic noise source in DNN training to regularize the DNN model, thus making it adaptive to such non-ideal effect. As for thermal, shot, and random telegraph noise, we investigated to optimize system frequency to eliminate its accuracy degradation. Our proposed comprehensive methods together could effectively mitigate the non-ideal effects of the ReRAM crossbar and provide great potential for future large scale accurate DNN crossbar based accelerator.

# Chapter 4

# INPUT SECURITY OF NEURAL NETWORK

In addition to the efficiency perspective elaborated in the prior chapters, the security-related issues of the neural network serving are non-trivial, which should be carefully handled as well. One of the emerging topics called adversarial attack [33] has attracted a significant amount of research efforts, owing to the damage it could potentially cause. Adversarial attack normally refers to adversary maliciously perturbs the DNN input natural data with human imperceptible noise, but causes the malfunction of neural network with erroneous prediction. In contrast to the plentiful investigations of the adversarial attack, there are still demanding more effective adversarial defense techniques, to improve DNN resistance against adversarial attacks. In this chapter, we will focus on developing an effective adversarial defense method via leveraging the trainable noise injection. Note that, to distinguish an adversarial attack variant from the following chapter, all the adversarial attacks discussed in the chapter are adversarial attacks on input data.

## 4.1 Preliminaries

### 4.1.1 Prior Adversarial Attacks

Recently, various powerful adversarial attack methods have been proposed to fool a trained DNN through introducing barely visible perturbation upon input data. Several state-of-the-art white-box (i.e., PGD [33], FGSM [31] and C&W [45]) and black-

box (i.e., Substitute [47] and ZOO [44]) adversarial attack methods, which will be investigated in this work, are briefly introduced as follows.

**FGSM Attack.** Fast Gradient Sign Method (FGSM) [31], [32] is an efficient single-step adversarial attack method. Given vectorized input $\boldsymbol{x}$ and corresponding target label $\boldsymbol{t}$, FGSM alters each element $x$ of $\boldsymbol{x}$ along the direction of its gradient w.r.t the inference loss $\partial\mathcal{L}/\partial x$. The generation of adversarial example $\hat{\boldsymbol{x}}$ (i.e., perturbed input) can be described as:

$$\hat{\boldsymbol{x}} = \boldsymbol{x} + \epsilon \cdot \text{Sign}\big(\nabla_{\boldsymbol{x}}\mathcal{L}(g(\boldsymbol{x};\boldsymbol{\theta}), \boldsymbol{t})\big) \tag{4.1}$$

where $\epsilon$ is the perturbation constraint that determines the attack strength. $g(\boldsymbol{x};\boldsymbol{\theta})$ computes the output of DNN paramterized by $\boldsymbol{\theta}$. $\text{Sign}(\cdot)$ is the sign function. Note that, the attack is followed by a clipping operation to ensure the $\hat{x} \in [0, 1]$.

**PGD Attack.** Projected Gradient Descent (PGD) [33] is a multi-step variant of FGSM, which is one of the strongest $L^\infty$ adversarial example generation algorithm. With $\hat{\boldsymbol{x}}^{k=1} = \boldsymbol{x}$ as the initialization, the iterative update of perturbed data $\hat{\boldsymbol{x}}$ in $k$-th step can be expressed as:

$$\hat{\boldsymbol{x}}^k = \Pi_{P_\epsilon(\boldsymbol{x})}\Big(\hat{\boldsymbol{x}}^{k-1} + a \cdot \text{Sign}\big(\nabla_{\boldsymbol{x}}\mathcal{L}(g(\hat{\boldsymbol{x}}^{k-1};\boldsymbol{\theta}), t)\big)\Big) \tag{4.2}$$

where $P_\epsilon(\boldsymbol{x})$ is the projection space which is bounded by $\boldsymbol{x} \pm \epsilon$, and $a$ is the step size. Madry et al. [33] also propose that PGD is a universal adversary among all the first-order adversaries (i.e., attacks only rely on first-order information).

**C&W Attack.** Carlini and Wagner propose an attack method called C&W attack [45]. C&W attack considers the generation of adversarial example as a problem of optimizing the $L^p$-norm of distance metric $\delta$ w.r.t the given input data $\boldsymbol{x}$, which can be described as:

$$||\boldsymbol{\delta}||_p = \Big(\sum_{i=1}^{n} |\delta_i|^p\Big)^{1/p}; \quad \delta_i = \hat{x}_i - x_i \tag{4.3}$$

$$\text{minimize}||\boldsymbol{\delta}||_p + c \cdot \mathcal{L}(\boldsymbol{x} + \boldsymbol{\delta}) \quad s.t. \quad \boldsymbol{x} + \boldsymbol{\delta} \in [0,1]^n \tag{4.4}$$

where $\boldsymbol{\delta}$ is taken as the perturbation added upon the input $\boldsymbol{x}$, and a specific loss function $\mathcal{L}$ is chosen in [45] to solve the optimization problem via gradient descent. $c$ is a constant set by the attacker. In this work, we use $L^2$-norm based C&W attack and take $||\boldsymbol{\delta}||_{p=2}$ as the evaluation metric to measure the robustness of DNN, where a greater value of $||\boldsymbol{\delta}||_{p=2}$ normally indicates a DNN possesses higher robustness against potential adversarial attacks.

**Black-box Attacks.** The most popular black-box attack is conducted using a substitute model [47], which is trained using the output of the target model as the label, to mimic the functionality of the target model. Then, the adversarial example generated from the substitute model is used to perform the attack on the target model. In this work, we specifically investigate the transferable adversarial attack [46], which is a variant of substitute model attack [47]. In the transferable adversarial attack, the adversarial example is generated from one source model to attack another target model. The source model and target can own a totally different structure but trained with the real training data. Beyond that, Zero-th Order Optimization (ZOO) attack [44] is investigated in this work as well. Rather than using the substitute model to approximate the gradients of the target model to perform an attack, the ZOO attack directly approximates the gradient based on the input data and output scores using a stochastic gradient coordinate.

### 4.1.2 Prior Defenses

Improving network robustness via adversarial training [32], [33] is by far the most popular and unbroken defense approach. The key idea of adversarial training is to

take the adversarial example as the training data, which trains the DNN against the adversarial attack. Most of the later works [106], [107] have followed this path to supplement their defense with adversarial training. The initial and important step in adversarial training is to choose an attack model for adversarial example generation. Adopting Projected Gradient Descent (PGD) [33] as an attack model for adversarial training is becoming popular, since it is considered to be capable of generating universal adversarial examples among the first-order approaches [33]. Additionally, among many recent defense methods, only PGD based adversarial training can sustain the state-of-the-art accuracy under various other attacks [32], [45], [108].

Recent work [109] has merged the concept of improving model robustness through regularization to defend adversarial examples. A well-known method for model regularization is noise injection, which is a variant of dropout on weights [110] or activation [111]. For further improving the performance of DNN under attack, there are works attempt to introducing randomness into DNN for adversarial defense, such as randomly pruning some activation during the inference [112], randomizing the input layer [113], inserting a noise-layer right before the convolution layers [49], [50]. However, the performance improvement (i.e., perturbed-data accuracy) mainly comes from the stochastic gradient instead of regularizing the DNN for better robustness, which is considered as the broken defense method according to the criterion of gradient obfuscations [108]. An alternative and straight-forward approach to evaluating adversarial defense about the gradient obfuscation is to examine the clean- (attack free) and perturbed-data (under attack) accuracy. If the adopted method mainly performs the model regularization, it is expected to improve the perturbed-data accuracy without sacrificing the clean-data accuracy.

Last but not least, we also notice that recent work Adv-BNN [48] also combines the adversarial training and noise injection on weight (i.e., Bayesian neural network equivalently). In comparison to our proposed PNI, Adv-BNN [48] mainly has the following drawbacks: 1) Significant computational and storage overhead owing to the used weight posterior (double model size) and output ensemble (>10 times), and 2) potential gradient obfuscation (trade the clean-data accuracy with perturbed-data accuracy). The key factor that our PNI outperforms Adv-BNN is the layer-wise noise injection (Eq. (4.5)) and ensemble loss function (Eq. (4.10)), which will be explicitly introduced in the following sections.

## 4.2 Parametric Noise Injection

In this section, we first introduce the proposed Parametric Noise Injection (PNI) function and will investigate the impact of noise injection on input/weight/activation.

### 4.2.1 Definition

The method proposed here to inject Gaussian noise to different components or locations within DNN can be mathematically described as:

$$\tilde{v}_{l,i} = f_{\text{PNI}}(v_{l,i}) = v_{l,i} + \alpha_l \cdot \eta_{l,i}; \quad \eta_{l,i} \sim \mathcal{N}(0, \sigma_l^2) \tag{4.5}$$

where $v_{l,i}$ is the element of noise-free tensor $\boldsymbol{v}_l$ in $l$-th layer of DNN, and such $\boldsymbol{v}_l$ can be input/weight/inter-layer (i.e., activation) tensor in this work. $\eta_{l,i}$ is the noise term which samples from Gaussian distribution with zero mean and variance $\sigma_l^2$, for each inference. $\alpha_i$ is the coefficient that scales the magnitude of injected noise $\eta_l$. Note that,

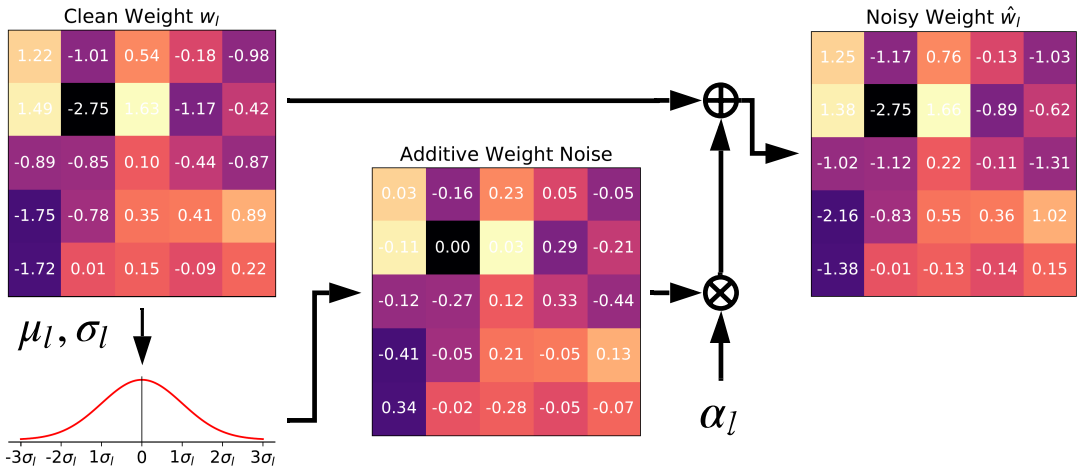



Figure 17: The flowchart of Parametric Noise Injection (PNI) on the weight matrix $\boldsymbol{W}_l$ of a $5 \times 5$ fully-connected layer (i.e., PNI-W). For each inference, the process of PNI on $\boldsymbol{W}_l$ can be divided into three sequential steps: 1) Statistically calculate the standard deviation $\sigma_l$ of $\boldsymbol{W}_l$; 2) Sample the additive weight noise (i.i.d) from $\mathcal{N}(0, \sigma_l^2)$; 3) Scale the run-time sampled weight noise with a trainable $\alpha_l$, then the noise weight is obtained via applying the additive weight noise upon the clean weight.

we adopt the scheme that $\eta_l$ shares the identical variance with $\boldsymbol{v}$ as in Eq. (4.5), thus the injected additive noise relies on $\alpha_l$ and the distribution of $\boldsymbol{v}_l$ simultaneously.

In this work, rather than manually configuring $\alpha_l$ to restrict the noise level, we set $\alpha_l$ as a learnable parameter that can be optimized for network robustness improvement. We name such a method as *Parametric Noise Injection* (PNI). Assuming we perform the proposed PNI on the weight tensors of convolution/fully-connected layers throughout the entire DNN, for each parametric layer there is only one layer-wise noise scaling coefficient ($\alpha_l$) to be optimized. We take such layer-wise PNI configuration as default in this work. An example of PNI on weight (i.e., PNI-W) is depicted in Fig. 17.

### 4.2.2 Optimization

In this work, we treat the noise scaling coefficient as a model parameter which can be optimized through the back-propagation. For $f_{\text{PNI}}(\cdot)$ that shares the noise scaling coefficient layer-wise, the gradient computation can be described as:

$$\frac{\partial \mathcal{L}}{\partial \alpha_l} = \sum_i \frac{\partial \mathcal{L}}{\partial f_{\text{PNI}}(v_{l,i})} \frac{\partial f_{\text{PNI}}(v_{l,i})}{\partial \alpha_l} \tag{4.6}$$

where the $\sum_i$ takes the summation over the entire tensor $\boldsymbol{v}_{l,i}$, and $\partial \mathcal{L} / \partial f_{\text{PNI}}(v_{l,i})$ is the gradient back-propagated from the followed layers. The gradient calculation of the PNI function is:

$$\frac{\partial f_{\text{PNI}}(v_{l,i})}{\partial \alpha_l} = \eta_{l,i} \tag{4.7}$$

It is noteworthy that the random sampled $\eta_{l,i}$ will be taken as a constant during the back-propagation. Using the gradient descent optimizer with momentum, the optimization of $\alpha$ at step $j$ can be written as:

$$V_l^j = m \cdot V_l^{j-1} + \frac{\partial \mathcal{L}^{j-1}}{\partial \alpha_l}; \quad \alpha_l^j = \alpha_l^{j-1} - \epsilon \cdot V_l^j \tag{4.8}$$

where $m$ is the momentum, $\epsilon$ is the learning rate, and $V_l$ is the updating velocity. Moreover, since weight decay tends to make the learned noise scaling coefficient converge to zero, there is no weight decay term on the $\alpha$ during the parameter updating in this work. We set $\alpha_l = 0.25$ as default initialization.

### 4.2.3 Robust Optimization

We expect to utilize the aforementioned PNI technique to improve the network's robustness. However, directly optimizing the noise scaling coefficient normally leads $\alpha_l$ to converge at a small close-to-zero value (vanilla training in Table 15), owing to the

gradient-descent optimizer tends to make the weights to be noise-free thus to over-fit the training data.

In order to succeed in adversarial defense, we jointly use the PNI method with robust optimization (a.k.a. *Adversarial Training*) which can boost the inference accuracy for the perturbed data under attack. Given inputs- $\boldsymbol{x}$ and target labels- $t$, the adversarial training is to obtain the optimal solution of network parameter $\boldsymbol{\theta}$ for the following min-max problem:

$$\min_{\boldsymbol{\theta}} \Big\{ \max_{\boldsymbol{x}' \in P_\epsilon(\boldsymbol{x})} \mathcal{L}\big(g(\hat{\boldsymbol{x}}; f_{\text{PNI}}(\boldsymbol{\theta})), \boldsymbol{t}\big) \Big\} \tag{4.9}$$

where the inner maximization tends to acquire the perturbed data $\hat{\boldsymbol{x}}$, and $P_\epsilon(\boldsymbol{x})$ is the input data perturb set constrained by $\epsilon$. While the outer minimization is optimized through gradient descent method as regular network training. $L^\infty$ PGD attack [33] is adopted as the default inner maximization solver (i.e., generating $\hat{\boldsymbol{x}}$).

Moreover, in order to balance the clean data accuracy and perturbed data accuracy for practical application, rather than performing the outer minimization solely on the loss of perturbed data as in Eq. (4.9), we minimize the ensemble loss $\mathcal{L}'$ which is the weighted sum of losses for clean- and perturbed-data. The ensemble loss $\mathcal{L}_{\text{ens}}$ is described as:

$$\mathcal{L}_{\text{ens}} = w_c \cdot \mathcal{L}(g(\boldsymbol{x}; f_{\text{PNI}}(\boldsymbol{\theta})), \boldsymbol{t}) + w_a \cdot \mathcal{L}(g(\hat{\boldsymbol{x}}; f_{\text{PNI}}(\boldsymbol{\theta})), \boldsymbol{t}) \tag{4.10}$$

where $w_c$ and $w_a$ are the weights for clean data loss term and adversarial data loss term, respectively. $w_c = w_a = 0.5$ is the default configuration in this work.

Optimizing the ensemble loss $\mathcal{L}_{\text{ens}}$ is the key to the successful training of both the model's inherent parameter (e.g. weight, bias) and the add-on noise scaling coefficient $\alpha_l$ from PNI. The intuition behind is that the gradient-descent optimizer attempt to find an equilibrium point of $\alpha_l$ when minimizing $\mathcal{L}_{\text{ens}}$. If $\alpha_l$ is too large, PNI will introduce significant noise into the inference path which will definitely hamper the

accuracy for both clean- and perturbed-data. If $\alpha_l$ is too small, PNI will not perform any regularization.

## 4.3 Experiment

### 4.3.1 Experiment Setup

#### 4.3.1.1 Datasets and Network Architectures

Two visual datasets for object recognition task is considered in this work, which is MNIST and CIFAR-10. The MNIST [103] dataset is a set of handwritten digit 28×28 gray-scale images with 60K training examples and 10K test examples. The CIFAR-10 [104] dataset is composed of 50K training samples and 10K test samples of 32×32 color image. There is no data augmentation used for MNIST, while CIFAR-10 uses the same augmentation method as in [2]. Although we test our method on both CIFAR-10 and MNIST, we mainly present results on CIFAR-10 to validate our method. Since the results on MNIST cannot provide much insight, we put the MNIST results in the appendix.

For MNIST, we test the performance using the variant LetNet5[10]. For CIFAR-10, the classical Residual Networks [2] (ResNet-20/32/44/56) architecture are used, and ResNet-20 is taken as the baseline for most of the comparative experiments and ablation studies. A redundant network ResNet-18 is also used to report the performance for CIFAR-10, since large network capacity is helpful for adversarial defense. Moreover, rather than including the input normalization within the data augmentation, we place

[10]https://github.com/pytorch/examples/blob/master/mnist/

a non-trainable data normalization layer in front of the DNN to perform the identical function, thus an attacker can directly add the perturbation on the natural image. Note that, since both PNI and PGD attack [33] include randomness, we report the accuracy in the format of mean±std% with 5 trials to alleviate error.

#### 4.3.1.2 Adversarial Attacks

To evaluate the performance of our proposed PNI technique, we employ multiple powerful white-box and black-box attacks as introduced in Section 4.1.1. For PGD attack on MNIST and CIFAR-10, $\epsilon$ is set to 0.3/1 and 8/255, and $N_{\text{step}}$ is set to 40 and 7 respectively. FGSM attack adopts the same $\epsilon$ setup as PGD. The attack configurations of PGD and FGSM are identical as the setup in [33], [106]. For the C&W attack, we set the constant $c$ as 0.01. ADAM [114] is used to optimize the Eq. (4.4) with learning rate as $5e^{-4}$. We choose 0 for the confidence coefficient of $k$, which is defined in the loss function used by the C&W $L^2$ attack in [45]. The binary search steps for the attack is 9, while the number of iteration to perform the gradient descent is 10. Moreover, we also conduct the PNI defense against several state-of-the-art black-box attacks (i.e. substitute [47], ZOO [44] and transferable [46] attack) in Section 4.3.2.2 to examine the robustness improvement resulted from the proposed PNI technique.

#### 4.3.1.3 Competing Methods for Adversarial Defense

As far as we know, the adversarial training with PGD [33] is the only unbroken defense method [108], which is labeled as *vanilla adversarial training* and taken as the baseline in this work. Beyond that, several recent works utilizing a similar concept as

ours in their defense method are discussed as well, including certified robustness [49], random self-ensemble [50], and Adv-BNN [48].

### 4.3.2 PNI for Adversarial Attacks

#### 4.3.2.1 PNI Against White-box Attacks

**Optimization method of PNI.** As the discussion at the end of Section 4.2.3, the noise scaling coefficient will not be properly trained without utilizing the adversarial training and ensemble loss. We conduct the experiments for training the layer-wise PNI on weight (PNI-W) of ResNet-20, to compare the convergence of trained noise. As tabulated in Table 15, simply performing the vanilla training using momentum SGD optimizer fails the adversarial defense, where the noise scaling coefficients $\alpha$ are converged to the negligible values. On the contrary, with the aid of adversarial training (i.e., optimization of Eq. (4.10)), convolution layers in the network's front-end have obtained relatively large $\alpha$ which are the bold values in Table 15, and the corresponding evolution curve are shown in Fig. 18.

Since the PGD attack [33] is taken as the inner maximization solver, the generation of adversarial example $\hat{\boldsymbol{x}}$ in Eq. (4.2) is reformatted as:

$$\hat{\boldsymbol{x}}^{t+1} = \Pi_{P_\epsilon(\boldsymbol{x})}\Big(\hat{\boldsymbol{x}}^t + a \cdot sgn\big(\nabla_{\boldsymbol{x}}\mathcal{L}(g(\hat{\boldsymbol{x}}^t; f_{\text{PNI}}(\boldsymbol{\theta})), \boldsymbol{t})\big)\Big) \tag{4.11}$$

where the difference between Eq. (4.2) and Eq. (4.11) is with/without PNI in $\hat{\boldsymbol{x}}$ generation. It is noteworthy that, keeping the noise term in the model for both adversarial example generation (Eq. (4.11)) and model parameter update is also the critical factor for the PNI optimization with adversarial training. Increasing noise level enhances the

Table 15: Convergence of Parametric Noise Injection (PNI): ResNet-20 with Layer-wise weight PNI on CIFAR-10 dataset. (Top) The converged layer-wise noise scaling coefficient $\alpha$ under various training scheme. (Bottom) Test accuracy for clean- and perturbed-data under PGD and FGSM attack.

| Layer Index | Vanilla Training | PNI-W+Adv. Train. (without PNI in $\hat{\boldsymbol{x}}$ generation) | PNI-W+Adv. Train. (with PNI in $\hat{\boldsymbol{x}}$ generation) |
|---|---|---|---|
| Conv0 | 0.003 | 0.004 | **0.146** |
| Conv1.0 | 0.002 | 0.005 | **0.081** |
| Conv1.1 | 0.004 | 0.004 | **0.049** |
| Conv1.2 | 0.002 | 0.001 | **0.097** |
| Conv1.3 | 0.004 | 5.856 | **0.771** |
| Conv1.4 | 0.005 | 0.005 | 0.004 |
| Conv1.5 | 0.002 | 0.001 | 0.006 |
| Conv2.0 | 0.004 | 0.000 | 0.006 |
| Conv2.1 | 0.006 | 0.003 | 0.004 |
| Conv2.2 | 0.004 | 0.003 | 0.030 |
| Conv2.3 | 0.001 | 0.006 | 0.003 |
| Conv2.4 | 0.003 | 0.001 | 0.033 |
| Conv2.5 | 0.002 | 0.001 | 0.023 |
| Conv3.0 | 0.007 | 0.001 | 0.008 |
| Conv3.1 | 0.003 | 0.001 | 0.006 |
| Conv3.2 | 0.007 | 0.002 | 0.001 |
| Conv3.3 | 0.006 | 0.001 | 0.002 |
| Conv3.4 | 0.009 | 0.002 | 0.001 |
| Conv3.5 | 0.005 | 0.000 | 0.001 |
| FC | 0.002 | 0.002 | 0.001 |
| Clean | 92.11% | 71.00% | 84.89±0.11% |
| PGD | 0.00±0.00% | 18.11% | 45.94±0.11% |
| FGSM | 14.08% | 26.34% | 54.48±0.44% |

defense strength, but hampers the network inference accuracy for natural clean image. Lowering $\alpha$, however, makes the network vulnerable to adversarial attack. As listed in Table 15, not incorporating the PNI-W in $\hat{\boldsymbol{x}}$ generation indeed leads to the failure of

Table 16: Effect of PNI location: The ResNet-20 clean- and perturbed-data (under PGD and FGSM attack) accuracy (mean±std%) on CIFAR-10 test-set, with PNI technique on different network location. Baseline is the ResNet-20 with vanilla adversarial training, and all the PNI combinations are optimized through adversarial training by default.

| | Test with PNI | | | Test without PNI | | |
|---|---|---|---|---|---|---|
| | Clean | PGD | FGSM | Clean | PGD | FGSM |
| Vanilla adv. train | - | - | - | 83.84 | 39.14±0.05 | 46.55 |
| PNI-W | 84.89±0.11 | **45.94±0.11** | **54.48±0.44** | 85.48 | 31.45±0.07 | 42.55 |
| PNI-I | 85.10±0.08 | 43.25±0.16 | 50.78±0.16 | 84.82 | 34.87±0.05 | 44.07 |
| PNI-A-a | 85.22±0.18 | 43.83±0.10 | 51.41±0.08 | 85.20 | 33.93±0.05 | 44.32 |
| PNI-A-b | 84.66±0.16 | 43.63±0.20 | 51.26±0.09 | 83.97 | 33.53±0.05 | 43.37 |
| PNI-W+A-a | 85.12±0.10 | 43.57±0.12 | 51.15±0.21 | 84.88 | 33.23±0.05 | 43.59 |
| PNI-W+A-b | 84.33±0.11 | 43.80±0.19 | 51.14±0.07 | 84.42 | 33.30±0.05 | 43.43 |

PNI optimization, and the large value ($\alpha = 5.856$ in Table 15) is not converged due to the probable gradient explosion.

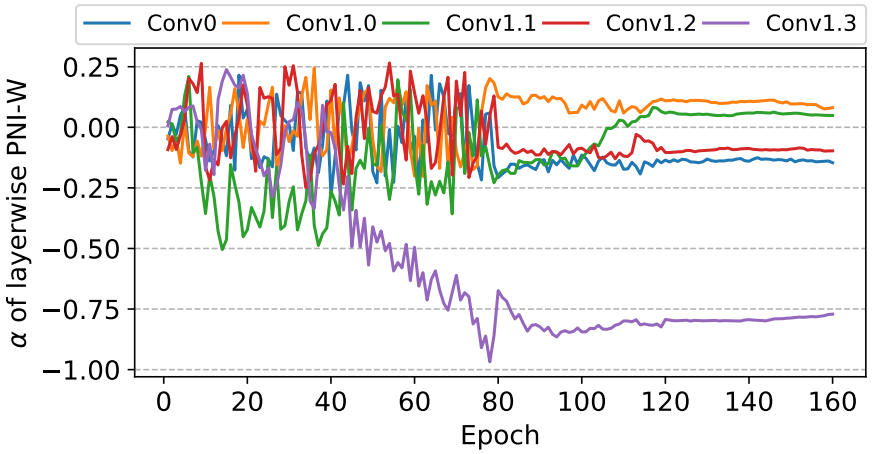



Figure 18: The evolution curve of trainable noise scaling coefficient $\alpha$ for layerwise PNI on weight (PNI-W). Only front 5 layers of ResNet-20 are shown. The learning rate of SGD optimizer is reduced at 80 and 120 epoch. Best viewed in color.

**Effect of PNI on weight, activation and input.** In this work, even though the scheme of injecting noise on the weight (PNI-W) is taken as the default PNI setup, more results about PNI on activation (PNI-A-a/b), input (PNI-I) and hybrid-mode (e.g. PNI-W+A) are provided in Table 16 for a comprehensive study. PNI-A-a/PNI-A-b denotes injecting noise on the output/input tensor of the convolution/fully-connected layer respectively. Moreover, PNI-A-b scheme intrinsically includes the PNI-I, since PNI-I is applying the noise on the input tensor of the first layer. Note that, all models with PNI variants are jointly trained with PGD-based adversarial training [33] as discussed above. Then, with the same trained model, we report the accuracy with/without the trained noise term (left/right in Table 16) during the test phase. As shown in Table 16, with the noise term enabled during the test phase, PNI-W on ResNet-20 gives the best performance to defend PGD and FGSM attack, in comparison to PNI on other locations. Although it is elusive to fully understand the mechanism that PNI-W outperforms other counterparts, the intuition is that PNI-W is the generalization of PNI-A in each connection instead of each output unit, similar as the relation between the regularization technique DropConnect [110] and Dropout [111].

Furthermore, we also observe that disabling PNI during the test phase leads to significant accuracy drop for defending PGD and FGSM attack, while the clean-data accuracy maintains the same level as PNI enabled. Such observation raises two concerns about our PNI techniques: 1) Does the improvement of clean-/perturbed-data accuracy with PNI mainly comes from the attack strength reduction caused by the randomness (potential gradient obfuscation [108])? 2) Is PNI just a negligible trick or it performs the model regularization to construct a more robust model? Our answers to both questions are negative, where the explanations are elaborated under Section 4.4.

**Effect of network capacity.** In order to investigate the relation between network capacity (i.e., number of trainable parameters) and robustness improvement by PNI, we examine various network architectures in terms of both depth and width. For different network depths, experiments on ResNet-20/32/44/56 [2] are conducted under vanilla adversarial training [33] and our proposed PNI robust optimization method. For different network widths, we adopt the original ResNet-20 as a baseline and expand its input&output channel of each layer by 1.5×/2×/4× respectively. Same as Table 16, we report clean- and perturbed-data accuracy with/without PNI term during the test phase. The results in Table 18 indicates that increasing the model's capacity indeed improves network robustness against white-box adversarial attacks, and our proposed PNI outperforms vanilla adversary training in terms of both clean-data accuracy and perturbed data accuracy for PGD and FGSM attack. Such observation demonstrates that the perturbed-data accuracy improvement does not come from trading off clean-data accuracy as reported in [106], [115]. Through increasing the network capacity, the robustness improvement results from the proposed PNI becomes less significant. Although both adversarial training and PNI techniques perform regularization, the network structure still needs careful construction to prevent over-fitting resulted from over-parameterization.

Table 17: C & W attack $L_2$-norm comparison

| | | C&W $L_2$-norm | | |
|---|---|---|---|---|
| Model | capacity | No defense | Vanilla adv. train | PNI-W |
| ResNet-20 (4×) | 4,286,026 | 0.12 | 1.97 | 1.95±0.02 |
| ResNet-18 | 11,173,962 | 0.12 | 2.39 | 2.62±0.04 |

**Robustness evaluation with C&W attack.** Improved robustness does not necessarily mean improving the test data accuracy against any particular attack method.

Table 18: Effect of network depth and width: The clean- and perturbed-data (under PGD and FGSM attack) accuracy (mean±std%) on CIFAR-10 test-set, utilizing different robust optimization configurations. For network depth, the classical ResNet-20/32/44/56 with increasing depth is reported. For network width, the ResNet-20 (1×) is adopted as the baseline, then we compare the wide ResNet-20 with the input and output channel scaled by 1.5×/2×/4×. Capacity denotes the number of trainable parameters in the model.

| | | No defense | | | Vanilla adv. train | | | PNI-W+adv. train (Test with PNI) | | | PNI-W+adv. train (Test without PNI) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Model | Capacity | Clean | PGD | FGSM | Clean | PGD | FGSM | Clean | PGD | FGSM | Clean | PGD | FGSM |
| Net20 | 269,722 | 92.1 | 0.0±0.0 | 14.1 | 83.8 | 39.1±0.1 | 46.6 | 84.9±0.1 | 45.9±0.1 | 54.5±0.4 | 85.5 | 31.6±0.1 | 42.6 |
| Net32 | 464,154 | 92.8 | 0.0±0.0 | 17.8 | 85.6 | 42.1±0.0 | 50.3 | 85.9±0.1 | 43.5±0.3 | 51.5±0.1 | 86.4 | 35.3±0.1 | 45.5 |
| Net44 | 658,586 | 93.1 | 0.0±0.0 | 23.9 | 85.9 | 40.8±0.1 | 48.2 | 84.7±0.2 | 48.5±0.2 | 55.8±0.1 | 86.0 | 39.6±0.1 | 49.9 |
| Net56 | 853,018 | 93.3 | 0.0±0.0 | 24.2 | 86.5 | 40.1±0.1 | 48.8 | 86.8±0.2 | 46.3±0.3 | 53.9±0.1 | 87.3 | 41.6±0.1 | 51.1 |
| Net20(1.5×) | 605,026 | 93.5 | 0.0±0.0 | 15.9 | 85.8 | 42.0±0.0 | 49.6 | 86.0±0.1 | 46.7±0.2 | 54.5±0.2 | 87.0 | 38.4±0.1 | 49.1 |
| Net20(2×) | 1,073,962 | 94.0 | 0.0±0.0 | 13.0 | 86.3 | 43.1±0.1 | 52.6 | 86.2±0.1 | 46.1±0.2 | 54.6±0.2 | 86.8 | 39.1±0.0 | 50.3 |
| Net20(4×) | 4,286,026 | 94.0 | 0.0±0.0 | 14.2 | 87.5 | 46.1±0.1 | 54.1 | 87.7±0.1 | 49.1±0.3 | 57.0±0.2 | 88.1 | 43.8±0.1 | 54.2 |

Typically $L_2$-norm based C & W attack [45] should reach 100 % success rate against any defense. Thus average $L_2$ norm required to fool the network gives more insight about a network's robustness in general [45]. The result presented in Table 17 represents the overall performance of our model against C&W attack. Our method of training the noise parameter becomes more effective for a more redundant network. We demonstrate this phenomenon by performing a comparison study between ResNet-20 and ResNet-18 architecture. Clearly, ResNet-18 shows the improvement in robustness from Vanilla adv. training much more than ResNet-20 against C&W attack.

#### 4.3.2.2 PNI Against Black-box Attacks

In this section, we test our proposed PNI technique against the transferable adversarial attack [46] and ZOO attack. Following the transferable adversarial attack [46], two trained neural network is taken as the source model ($S$) and target model ($T$). The adversarial examples $\hat{\boldsymbol{x}}_s$ is generated from the source model then attack the target model using $\hat{\boldsymbol{x}}_s$, which is denoted as $S \Rightarrow T$. We take ResNet-18 on CIFAR-10 as an example. We train two ResNet-18 model (model-A and B) on CIFAR-10 dataset to attack each other, where model-A is optimized through vanilla adversarial training, while model-B is trained using our proposed PNI variants (i.e., PNI-W/A-a/W+A-a) robust optimization method. Table 19 shows almost equal perturbed-data accuracy for A $\Rightarrow$ B and B $\Rightarrow$ A under various PNI scenarios, which indicates that the presence of PNI during the inference has a negligible effect on the attack strength of PGD.

For ZOO attack [44], we test our defense on 200 randomly selected test samples for an untargeted attack. The Attack success rate denotes the percentage of test sample change their classification to the wrong class after the attack. ZOO attack success

Table 19: PNI against black-box attacks: On CIFAR-10 test subset, (Left) perturbed-data accuracy under transferable PGD attack, and (Right) the attack success rate for ZOO attack. Model-A is a ResNet-18 trained by vanilla adversarial training, and Model-B is a ResNet-18 trained by PNI-W/A-a/W+A-a with adversarial training.

| | Transferable attack | | ZOO attack |
|---|---|---|---|
| Train. scheme of B | A $\Rightarrow$ B | B $\Rightarrow$ A | success rate |
| PNI-W | 75.13$\pm$0.17 | 75.23$\pm$0.18 | 57.72 |
| PNI-A-a | 74.67$\pm$0.11 | 75.86$\pm$0.13 | 69.61 |
| PNI-W+A-a | 75.14$\pm$0.10 | 74.92$\pm$0.13 | 50.00 |

rate for vanilla ResNet-18 with adversarial training is close to 80 %. The robustness of PNI is more evident from Table 19 as the attack success rate drops significantly for PNI-W+A-a and PNI-W. However, PNI-A-a fails to resist the ZOO attack even though it still maintains a lower success rate than the baseline. The failure of PNI-A-a shows that just adding noise in-front of the activation does not necessarily achieve the desired robustness as claimed by some of the previous defenses [49], [50].

#### 4.3.2.3 Comparison to Competing Methods

As discussed in Section 4.1.2, a large number of adversarial defense works have been proposed recently, however, most of them are already broken by stronger attacks proposed in [108], [116]. As a result, in this work we choose to compare with the most effective one to date - PGD based adversarial training [33]. Additionally, we compare with other randomness-based works [48]–[50] in Table 20 for examining the effectiveness of PNI.

Previous defense works [106], [115] have shown a trade-off between clean-data accuracy and perturbed-data accuracy, where the perturbed-data accuracy improvement

Table 20: Comparison of state-of-the-art adversarial defense methods with clean- and perturbed-data accuracy on CIFAR-10 under PGD attack.

| Defense method | model | Clean | PGD |
|---|---|---|---|
| PGD adv. train [33] | ResNet-20 (4×) | 87 | 46.1±0.1 |
| DP [49] | 28-10 Wide ResNet (L=0.1) | 87.0 | 25 |
| RSE [50] | ResNext | 87.5 | 40 |
| Adv-BNN [48] | VGG-16 | 79.7 | 45.4 |
| PNI-W (this work) | ResNet-20 (4×) | 87.7±0.1 | 49.1±0.3 |

achieved at the cost of lowering the clean-data accuracy. It is worthy to highlight that **the proposed PNI improves both clean- and perturbed data accuracy under the white-box attack, in comparison to PGD-based adversarial training** [33]. Differential Privacy (DP) [49] is a similar method of utilizing noise injection at various locations in the network. Although their defense guarantees a certified defense, it does not perform well against $L_\infty$-norm based attacks (e.g., PGD and FGSM). Moreover, to achieve a higher level of certified defense, DP significantly sacrifices the clean-data accuracy as well. Another randomness-based approach is Random Self-ensemble (RSE) [50], which inserts noise-layer before all the convolution layer. Even though their defense performs well against the C&W attack but poor against strong PGD attack. Beyond that, both DP and RSE manually configure the noise level which is extremely difficult to find the optimal setup. Whereas, in our proposed PNI method, the noise level is determined by a trainable layer-wise noise scaling coefficient and distribution of weight at noise injected location. For Adv-BNN [48], besides the computational overhead and model size ($> 20$ times), our PNI also outperforms it in terms of performance on both clean- and perturbed-data.

## 4.4 Discussion

The defense performance improvement led by our proposed PNI does not come from the stochastic gradients. The stochastic gradient is considered to incorrectly approximate the true gradient based on a single sample. We try to show that PNI is not relying on the gradient obfuscation from two perspectives:

- Our proposed PNI method passes each inspection item proposed by [108] to identify gradient obfuscation.
- Under PGD attack, through increasing the attack steps, our PNI robust optimization method still outperforms vanilla adversarial training (certified as non-obfuscated gradients in [108]).

Table 21: Checklist of examining the characteristic behaviors caused by obfuscated and masked gradient for PNI.

| Characteristics to identify gradient obfuscation [108] | Pass | Fail |
|---|---|---|
| 1. One-step attack performs better than iterative attacks | ✓ | |
| 2. Black-box attacks are better than white-box attacks | ✓ | |
| 3. Unbounded attacks do not reach 100% success | ✓ | |
| 4. Random sampling finds adversarial examples | ✓ | |
| 5. Increasing distortion bound doesn't increase success | ✓ | |

**Inspections of gradient obfuscation.** The famous gradient obfuscation work [108] enumerates several characteristic behaviors as listed in Table 21 which can be observed when the defense method owns gradient obfuscation. Our experiments show that PNI passes each inspection item in Table 21.

For item.1, all the experiments in Table 16 and Table 18 report that FGSM attack (one-step) performs worse than PGD attack (iterative). For item.2, our black-box at-

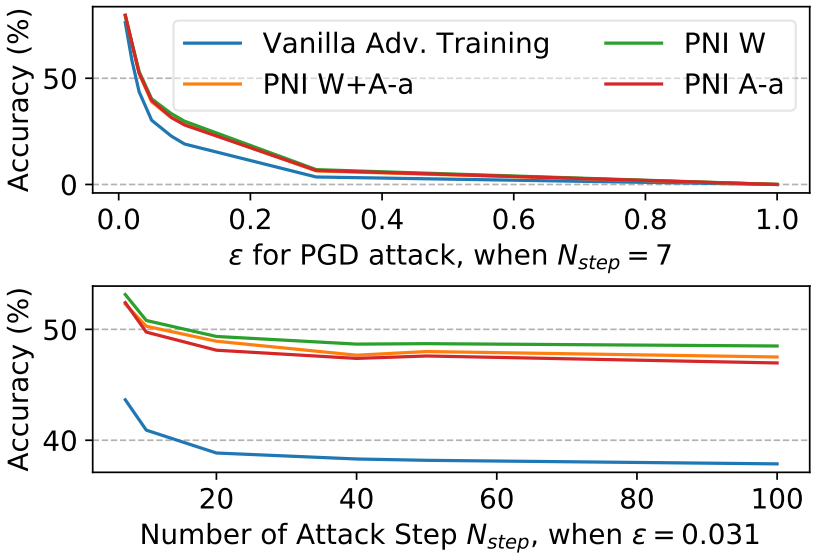



Figure 19: On CIFAR-10 test set, the perturbed-data accuracy of ResNet-18 under PGD attack (Top) versus attack bound $\epsilon$, and (Bottom) versus number of attack steps $N_{\text{step}}$

tack experiment in Table 19 shows that the black-box attack strength is worse than a white-box attack. For items.3, as plotted in Fig. 19, we run experiments through increasing the distortion bound-$\epsilon$. The result shows that the unbounded attacks do lead to 0% accuracy under attack. For item.4, the prerequisite is the gradient-based attack (e.g., PGD and FGSM) cannot find the adversarial examples, but the experiments in Fig. 19 reveals that our method still can be broken when increasing the distortion bound. It just increases the resistance against the adversarial attacks, in comparison to the vanilla adversarial training. For item.5, again as shown in Fig. 19, increasing the distortion bound increases the attack success rate.

**PNI does not rely on stochastic gradients.** As shown in Fig. 19, gradually increasing the PGD attack steps $N_{\text{step}}$ raises the attack strength [33], thus leading to perturbed-data accuracy degradation for both vanilla adversary training and our PNI technique. However, for both cases, the perturbed-data accuracy starts saturating and does not

degrade any further when $N_{\text{step}}$=40. If our PNI's success comes from the stochastic gradient which gives incorrect gradient owing to the single sample, increasing the attack steps suppose to eventually break the PNI defense which is not observed here. Our PNI method still outperforms vanilla adversarial training even when $N_{\text{step}}$ is increased up to 100. Therefore, we can draw the conclusion that, even if PNI does include gradient obfuscation, the stochastic gradient is not the dominant role in PNI for the robustness improvement.

## 4.5 Summary

In this chapter, the parametric noise injection technique is presented, where the noise intensity can be trained through solving the min-max optimization problem during adversarial training. Through extensive experiments, the proposed PNI method can outperform the state-of-the-art defense method in terms of both clean-data accuracy and perturbed-data accuracy.

# Chapter 5

# WEIGHT SECURITY OF NEURAL NETWORK

In the prior chapter, we have specifically discussed the adversarial attack and defense aspects with respect to the neural network input. As the DNN already shows vulnerability to the input, it inspires us to further explore the vulnerability of another DNN dimension – model parameters, especially model weights. Taken a compressed DNN with quantized weight into consideration, in this chapter, we will first conduct an investigation of adversarial attack upon weights, where a new adversarial weight attack is proposed. Followed by that, corresponding defense methods are proposed by us as well, to effectively enhance the resistance against the adversarial weight attack by a significant margin.

## 5.1 Preliminaries

**Memory bit-flip in real-world.** Flipping a memory cell bit within the memory system is a realistic and demonstrated threat model in existing computer systems. Recently, Kim *et al.*, [117] have demonstrated a method to cause memory bit-flip in DRAM merely through the frequent data accessing, which is now popularly known as Row-Hammer Attack (RHA). A malicious user can use RHA to modify the data stored in the DRAM memory cell by just flipping one bit at a time. [118] showed that by creating a profile for the bit flips in a DRAM, row hammer attack can effectively flip a single bit at any address in the software stack. According to the state-of-the-art investigations, common error detection and correction techniques, such as Error-Correcting Code

(ECC) [119] and Intel SGX [120], are broken defense mechanism to RHA. Such existing memory bit-flip attack (i.e., row-hammer attack) model brings a huge challenge to the security of DNN powered computing system since its parameters are normally stored in the main memory, i.e. DRAM, for maximizing the computation throughput, which is directly exposed to the adversarial attacker. Moreover, such a challenge becomes more severe because DNN powered applications are widely deployed in many resource-limited (e.g., smart IoT devices, mobile systems, edge devices, etc.) system that lacks the necessary data integrity check mechanism.

**Prior neural network parameter attack.** The adversarial example attack has been widely explored [121] to evaluate the robustness of DNN. However, we are still at the rudimentary stage towards investigating the effect of network parameter attack on neural network accuracy. Neural network parameters have been attacked using different levels of hardware Trojans, which require a specific pattern of input to trigger the Trojan inside the network [122], [123]. Moreover, such a Trojan attack requires hardware-level modifications, which may not be feasible in many practical applications. As a result, fault injection attacks could become a suitable alternative to attack DNN parameters [51]. For example, a Single Bias Attack (SBA) attacks a certain bias term of a neuron to change the classification of DNN to a different class [51]. Other works have injected faults into the activation function of the neural network to miss classify a target input [52].

**Limitations of Prior Works.** However, these previous attack algorithms are developed based on a full-precision model (i.e., network parameters are floating-point numbers stored in memory in the format of the IEEE standard for floating-point arithmetic [124]), where we believe such attack algorithms may not be efficient. Since it is extremely easy to cause DNN malfunction by just flipping the most significant

exponent bits of any random floating-point weight parameters. Through this simple method, it mainly causes DNN malfunction by exponentially increasing the magnitude of particular weight parameters by just several bit-flips. We conducted such an experiment to prove its efficiency in section 5.3.4. Based on our simulation results, it shows just 1 bit-flip of the most significant exponent bit of a random floating-point number weight could cause ResNet-18 network completely malfunction on the ImageNet dataset.

**Why we need a bit search algorithm?** On the other side, most of recent deep neural network applications are performed in a quantized platform such as google's Tensor Processing Unit (TPU) [125], that uses 8-bit operations for the quantized network. Such fixed precision models are more robust to network parameter perturbation. Similarly, we conducted another experiment to randomly choose a quantized weight for bit-flip attack using RHA. The simulation results in Fig. 24 show that 100 bit-flip in a quantized ResNet-18 could only cause 0.6% accuracy degradation in ImageNet, which indicates that random selection of quantized weight parameters to be attacked is not efficient and feasible. Thus, an efficient algorithm is required to search for the most vulnerable weights/bits in a quantized DNN.

## 5.2 Bit-Flip Based Adversarial Weight Attack

In this section, we present a novel Bit-Flip Attack (BFA) method to maliciously cause a DNN system malfunction through flipping an extremely small amount of vulnerable bits of weights. Our proposed algorithm, called Progressive Bit Search (PBS), is to identify those vulnerable DNN weight parameters (stored in terms of memory bits in DRAM) that could maximize the accuracy degradation with the minimum num-

ber of bit-flips. It is worth noting that this work focuses on BFA on a more robust DNN with quantized weight parameters instead of floating-point number weights as discussed earlier.

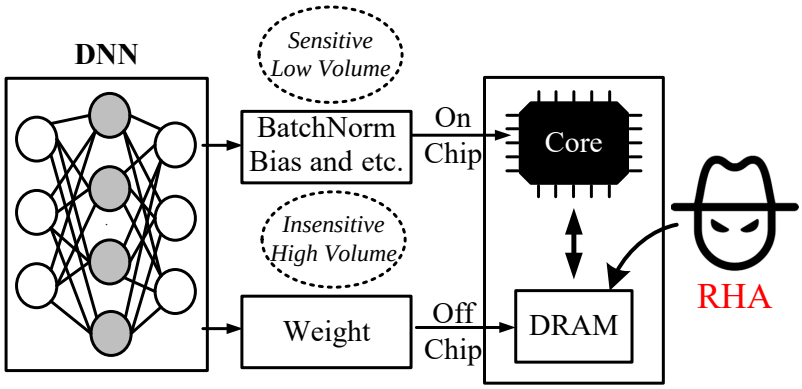



Figure 20: Concept illustration of quantized DNN under BFA.

### 5.2.1 Problem Definition of BFA

Given a quantized DNN contains $L$ convolutional/fully-connected layers, the original weights in floating-point are symmetrically quantized into $2^{N_q}-1$ levels with $N_q$-bits uniform quantizer. The quantized weights $\mathbf{W}$ are arithmetically represented in $N_q$-bits signed integer. In the computing memory system, $\mathbf{W}$ is stored in the format of twos complement[11], which is denoted as $\mathbf{B}$ in this chapter. More details of weights quantization are described in Section 5.2.2. The goal of this work is to find the optimal combination of vulnerable weight bits to perform BFA, thus maximizing the inference loss of DNN parameterized by the perturbed weights whose twos complement representation is $\hat{\mathbf{B}}$. Such vulnerable bit searching problem can be formulated as an

[11]All the binary weight mentioned hereinafter referred to as the weights in twos complement.

Table 22: Threat model of Bit-Flip Attack (BFA).

| Access Required ✓ | Access **NOT** Required ✗ |
|---|---|
| Model topology & parameters | Hyper-parameters and other training configurations. |
| A mini-batch of sample data | Complete train/test datasets. |

optimization problem as:

$$\max_{\{\hat{\mathbf{B}}_l\}} \mathcal{L}\Big(f\big(\boldsymbol{x}; \{\hat{\mathbf{B}}_l\}_{l=1}^{L}\big), \boldsymbol{t}\Big) - \mathcal{L}\Big(f\big(\boldsymbol{x}; \{\mathbf{B}_l\}_{l=1}^{L}\big), \boldsymbol{t}\Big)$$
$$\text{s.t.} \sum_{l=1}^{L} \mathcal{D}(\hat{\mathbf{B}}_l, \mathbf{B}_l) \in \{0, 1, ..., N_{\text{b}}\} \tag{5.1}$$

where $\boldsymbol{x}$ and $\boldsymbol{t}$ are the vectorized input and target output[12]. Taken $\boldsymbol{x}$ as the input, the inference computation of network parameterized by $\{\hat{\mathbf{B}}_l\}_{l=1}^{L}$ is expressed as $f(\boldsymbol{x}; \{\hat{\mathbf{B}}_l\}_{l=1}^{L})$. Note that $\mathcal{L}(\cdot, \cdot)$ calculates the loss between DNN output and target. $\mathcal{D}(\hat{\mathbf{B}}_l, \mathbf{B}_l)$ computes the Hamming distance between clean- and perturbed-binary weight tensor, and $N_{\text{b}}$ is maximum Hamming distance allowed through the entire DNN.

### 5.2.2 Weight Quantization and Encoding

**Weight Quantization.** in this chapter, we adopt a layer-wise $N_{\text{q}}$-bits uniform quantizer for weight quantization. For $l$-th layer, the quantization process from the floating-point base $\mathbf{W}_l^{\text{fp}}$ to its fixed-point (signed integer) counterpart $\mathbf{W}_l$ can be described as:

$$\Delta w_l = \max(\mathbf{W}_l^{\text{fp}})/(2^{N_{\text{q}}-1} - 1); \quad \mathbf{W}_l^{\text{fp}} \in \mathbb{R}^d \tag{5.2}$$

$$\mathbf{W}_l = \text{round}(\mathbf{W}_l^{\text{fp}}/\Delta w_l) \cdot \Delta w_l \tag{5.3}$$

[12]Note that, all the targets $\boldsymbol{t}$ in this chapter are not the ground-truth labels, but the outputs of the clean DNN w.r.t the input data.

where $d$ is the dimension of weight tensor, $\Delta w_l$ is the step size of weight quantizer. For training the quantized DNN with non-differential stair-case function (in Eq. (5.3)), we use the straight-through estimator [83] as other works [39]. Note that, since $\Delta w_l \in \mathbb{R}$ is the coefficient shared by all the weights in $l$-th layer, we only store its fixed-point part $(\mathbf{W}_l/\Delta w_l) \in \{-2^{N_q-1}, ..., 2^{N_q-1}\}^d$, rather than $\mathbf{W}_l$.

**Weight Encoding.** The computing system normally stores the signed integer in two's complement representation, owing to its efficiency in arithmetic operations (e.g., mul). Given one weight element $w \in \mathbf{W}_l$, the conversion from its binary representation ($\boldsymbol{b} = [b_{N_q-1}, ..., b_0] \in \{0, 1\}^{N_q}$) in two's complement can be expressed as:

$$w/\Delta w = g(\boldsymbol{b}) = -2^{N_q-1} \cdot b_{N_q-1} + \sum_{i=0}^{N_q-2} 2^i \cdot b_i \tag{5.4}$$

With the conversion relation described by $g(\cdot)$ in Eq. (5.4), we can inversely obtain the binary representation of weights $\mathbf{B}$ from its fixed-point counterpart as well.

### 5.2.3 Algorithm of BFA

In this chapter, we perform the BFA utilizing the similar mechanism as FGSM [31], which was used to generate adversarial example. The key idea of BFA is to flip the bits along its gradient ascending direction w.r.t the loss of DNN. We take the binary vector $\boldsymbol{b}$ in Eq. (5.4) as an example and attempt to perform BFA upon $\boldsymbol{b}$. We first calculates the gradients of $\boldsymbol{b}$ w.r.t loss as:

$$\nabla_{\boldsymbol{b}} \mathcal{L} = [\frac{\partial \mathcal{L}}{\partial b_{N_q-1}}, ..., \frac{\partial \mathcal{L}}{\partial b_0}] \tag{5.5}$$

where $\mathcal{L}$ is the inference loss of DNN parametrized by $\boldsymbol{b}$. The naive operation is to directly perform the bit-flip using the gradients obtained in Eq. (5.5) and get perturbed

Table 23: Truth table of Bit-Flip Attack (BFA). $b_i$ is the clean bit and $\hat{b}_i$ is the perturbed bit by BFA. $m$ indicates whether there exist value change between $b_i$ and $\hat{b}_i$. The positive and negative of $\partial\mathcal{L}/\partial b_i$ are represented by 1 and 0 respectively.

| $b_i$ | sign($\partial\mathcal{L}/\partial b_i$) | $\hat{b}_i$ | $m$ |
|---|---|---|---|
| 0 | 1 (+) | 1 | 1 |
| 0 | 0 (-) | 0 | 0 |
| 1 | 1 (+) | 1 | 0 |
| 1 | 0 (-) | 0 | 1 |

bits as:

$$\hat{\boldsymbol{b}} = \boldsymbol{b} + \text{sign}(\nabla_{\boldsymbol{b}}\mathcal{L}) \tag{5.6}$$

where $\text{sign}(\nabla_{\boldsymbol{b}}\mathcal{L}) \in \{-1, +1\}^{N_\text{q}}$. However, since the bit value is constrained between 0 and 1 ($\boldsymbol{b} \in \{0, 1\}^{N_\text{q}}$), flipping the bit as Eq. (5.6) could lead to data overflow. Ideally, the BFA is supposed to follow the truth table in Table 23. Thus, we mathematically redefine the BFA as follows:

$$\boldsymbol{m} = \boldsymbol{b} \oplus \big(\text{sign}(\nabla_{\boldsymbol{b}}\mathcal{L})/2 + 0.5\big) \tag{5.7}$$

$$\hat{\boldsymbol{b}} = \boldsymbol{b} \oplus \boldsymbol{m} \tag{5.8}$$

where $\oplus$ is the bit-wise xor operator. $\boldsymbol{m}$ is the mask which indicates whether to perform the bit-flip operation.

### 5.2.4 Progressive Bit Search

Rather than performing the BFA upon each bit throughout the entire network, our goal is to perform BFA in a more precise and effective fashion. In this subsection, we propose a method called Progressive Bit Search (PBS) which combines the gradient ranking and progressive search. The proposed PBS method attempts to identify and flip $n_\text{b}$ most vulnerable bits per BFA iteration ($n_\text{b} = 1$ by default), thus progressively

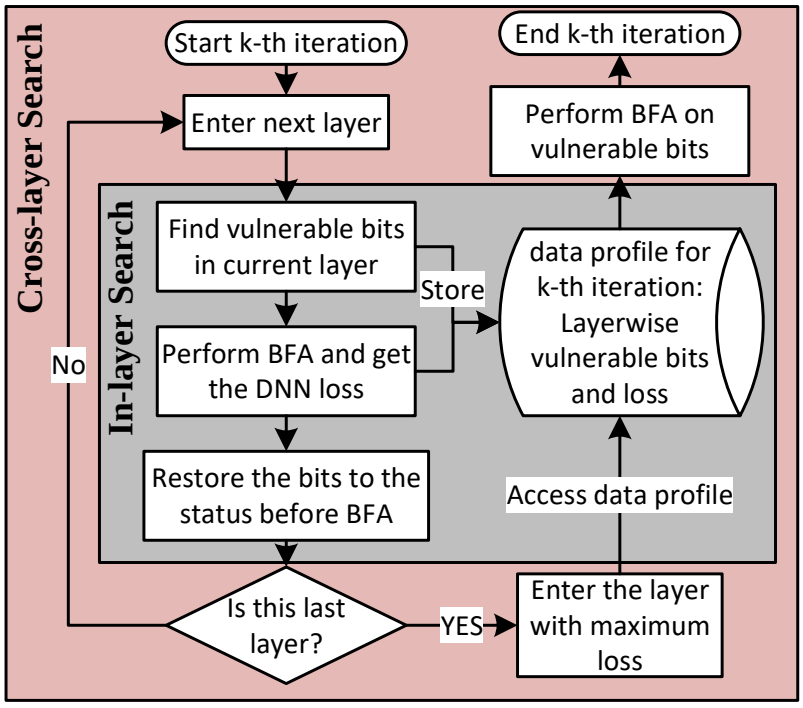



Figure 21: Flowchart to perform Progressive Bit Search (PBS) with in-layer and cross-layer search.

degrading the performance of DNN until it reaches the minimum accuracy or the preset number of iteration. As the flowchart of performing PBS depicted in Fig. 21, for each attack iteration, the process of bit searching can be generally divided into two successive steps: 1) **In-layer Search**: the in-layer search is performed through electing the $n_{\mathrm{b}}$ most vulnerable bits in the selected layer, then record the inference loss if those elected bits are flipped. 2) **Cross-layer Search**: with the in-layer search conducted upon each layer of the network independently, the cross-layer search is to evaluate the recorded loss increment caused by BFA with in-layer search, thus identify the top $n_{\mathrm{b}}$ vulnerable bits across different layers. The details of each step are described as follows.

**In-layer Search.** For the PBS in $k$-th iteration, in-layer searching of the $n_{\mathrm{b}}$ most vulnerable bits from $\hat{\mathbf{B}}_l^k$ in $l$-th layer is performed through gradient ranking. With the

given vectored input $\boldsymbol{x}$ and target $\boldsymbol{t}$, the inference and back-propagation are performed successively to calculate the gradients of bits w.r.t the inference loss. Then, we descendingly rank the vulnerability of bits by the absolute value of their gradients $\partial\mathcal{L}/\partial b$ and elect the bits whose gradients are top-$n_{\mathrm{b}}$, such process can be written as:

$$\hat{\boldsymbol{b}}_l^{k-1} = \underset{n_{\mathrm{b}}}{\mathrm{Top}}\left|\nabla_{\hat{\mathbf{B}}_l^{k-1}}\mathcal{L}\big(f(\boldsymbol{x};\{\hat{\mathbf{B}}_l^{k-1}\}_{l=1}^{L}),\boldsymbol{t}\big)\right| \tag{5.9}$$

where $\{\mathrm{Top}_{n_{\mathrm{b}}}\}$ function returns the pointer pointing at the storage of those elected $n_{\mathrm{b}}$ vulnerable bits. Then, we apply the BFA on those elected bits as:

$$\hat{\boldsymbol{b}}_l^{k} = \hat{\boldsymbol{b}}_l^{k-1} \oplus \boldsymbol{m} \tag{5.10}$$

where the mask $\boldsymbol{m}$ is generated following Eq. (5.7). Now, with the in-layer search and BFA performed on the $l$-th layer, we have to evaluate the loss increment caused by BFA in Eq. (5.10), which can be written as:

$$\mathcal{L}_l^k = \mathcal{L}\big(f(\boldsymbol{x};\{\hat{\mathbf{B}}_l^{k}\}_{l=1}^{L}),\boldsymbol{t}\big) \tag{5.11}$$

where the only difference between $\{\hat{\mathbf{B}}_l^{k}\}_{l=1}^{L}$ and $\{\hat{\mathbf{B}}_l^{k-1}\}_{l=1}^{L}$ are the bits flipped in Eq. (5.10). Note that, those bits flipped to $\hat{\boldsymbol{b}}_l^k$ in Eq. (5.10) will be restored back to $\hat{\boldsymbol{b}}_l^{k-1}$ after the loss evaluation is finished.

**Cross-layer Search.** As the aforementioned in-layer search can perform the layer-wise vulnerable bits election and BFA evaluation, the cross-layer search evaluates the BFA across the entire network. For the PBS in $k$-th iteration, the cross-layer search first independently conduct the in-layer search on each layer, and generate the loss set as $\{\mathcal{L}_1^k, \mathcal{L}_2^k, \cdots, \mathcal{L}_L^k\}$. Then, we could identify the layer-$j$ with maximum loss and re-perform the BFA (without restore) on the bits elected in $j$-th layer, which can be expressed as:

$$\begin{aligned} \hat{\boldsymbol{b}}_j^{k} &= \hat{\boldsymbol{b}}_j^{k-1} \oplus \boldsymbol{m} \\ \text{s.t.} j &= \arg\max_{l}\{\mathcal{L}_l^k\}_{l=1}^{L} \end{aligned} \tag{5.12}$$

After that, PBS is entered into $k+1$ iteration.

## 5.3 Experiments of BFA

### 5.3.1 Experiment Setup

**Datasets.** We take two visual datasets: CIFAR-10 [126] and ImageNet [30] for object classification task. CIFAR-10 contains 60K RGB images in size of $32 \times 32$. Following the standard practice, 50K examples are used for training and the remaining 10K for testing. The images are drawn evenly from 10 classes. ImageNet dataset contains 1.2M training images divided into 1000 distinct classes. The data augmentation used in this chapter is identical to methods in [2]. Note that, the proposed BFA is performed through randomly draw a sample of input images $\boldsymbol{x}$ from the test/validation set, where the default sample size is 128 and 256 for CIFAR-10 and ImageNet respectively. Then, only the sample input $\boldsymbol{x}$ is used to perform BFA, where the rest data and ground-truth labels are isolated from the attacker. Moreover, each experimental configuration is run with 5 trials to alleviate error caused by the randomness of sampling input $\boldsymbol{x}$.

**Network Architectures and quantization:** For CIFAR-10, experiments are conducted on a series of the residual networks (ResNet-20/32/44/56)[2], where the weights are quantized into 4/6/8 bit-width with retraining. For ImageNet, we choose a variety of famous network structures, including AlexNet, ResNet-18/34/50. Based on our observation, with high bit-width quantizer (e.g., $N_{\text{q}}$=8), directly quantizing the pre-trained full-precision DNN without retraining (i.e., fine-tuning) only shows negligible accuracy degradation. Therefore, for a fast evaluation of our proposed BFA on the

ImageNet dataset and its various network structures, we directly perform the weight quantization without retraining before conducting the BFA.

**Attack Formulation:** Traditional attacks mostly focus on attacking DNN by feeding perturbed inputs [31] to the network. Such adversarial attack can be grouped into two major categories: 1) white-box attack [31], [33], where the adversary has full access to the network architecture and parameters, and 2) black-box attack [44], [47], where the adversary can only access the input and output of a DNN without its internal configurations. For our proposed BFA, it demands full access to the DNN's weights and gradients. Thus BFA can be considered as a white-box attack. However, we assume that even under the white-box attack setup, the attacker has no access to the training dataset, training algorithm, and hyperparameters used during the training of the network.

### 5.3.2 BFA on CIFAR-10

Our bit-flip attack is evaluated across different architectures (i.e., ResNet-20/32/44/56) using varying quantized bit-widths (i.e., $N_q$=4/6/8) on CIFAR-10 dataset in Table 24. Without BFA, the quantized models show negligible accuracy degradation or even higher accuracy in comparison to their full-precision counterpart. The quantization noise introduced by the weight quantization is considered as a regularization method, which might contribute to the accuracy improvement when model training is over-fitting.

Since the CIFAR-10 dataset has 10 different classes of objects, degrading the model's accuracy down to 10% is equivalent to make the model as a random output generator. In contrast to adversarial example (e.g., PGD attack [33]), our proposed BFA is unable to degrade the network accuracy to 0%. The reason is the adversarial

Table 24: BFA on CIFAR-10 with ResNet-20/32/44/56, under various quantization bit-width ($N_{\text{q}}$=4/6/8). $N_{\text{flip}}$ is the number of bit-flips required (5 trials) to degrade the top-1 accuracy below 11% with BFA, regardless whether there exists bits flipped back to their original states. For CIFAR-10, top-1 accuracy with random guess is 10%. $D_{\text{B}}$ is the hamming distance between clean- and perturbed- binary weight ($D_{\text{B}} = \sum_{i=1}^{L} \mathcal{D}(\hat{\mathbf{B}}_l, \mathbf{B}_l)$). The bold number with underline highlight the mismatch between two corresponding $N_{\text{flip}}$ and $D_{\text{B}}$, which indicates there exist even bit-flips on the identical bit/bits.

| | Baseline Acc. | $N_{\text{q}} = 8$ | | | $N_{\text{q}} = 6$ | | | $N_{\text{q}} = 4$ | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Acc. | $N_{\text{flip}}$ | $D_{\text{B}}$ | Acc. | $N_{\text{flip}}$ | $D_{\text{B}}$ | Acc. | $N_{\text{flip}}$ | $D_{\text{B}}$ |
| Net20 | 92.11 | 92.28 | [7,10,10,12,17] | [7,10,10,12,17] | 91.89 | [8,8,11,12,13] | [8,8,11,12,13] | 91.85 | [7,7,7,8,12] | [7,7,7,8,12] |
| Net32 | 92.77 | 92.32 | [8,9,12,13,31] | [8,9,12,13,31] | 93.09 | [9,10,12,14,23] | [9,10,12,14,23] | 92.31 | [10,12,14,14,17] | [10,12,14,14,17] |
| Net44 | 93.10 | 93.60 | [6,10,11,13,22] | [6,10,11,13,22] | 93.39 | [13,13,15,16,17] | [13,13,15,16,17] | 91.52 | [14,14,15,16,50] | [14,14,15,16,50] |
| Net56 | 92.59 | 93.14 | [16,17,18,22,22] | [16,17,18,22,22] | 93.56 | [16,16,17,20,21] | [16,16,17,20,21] | 92.53 | [9,21,21,**23**,24] | [9,21,21,**21**,24] |

input attack is an input-specific attack that is designed to misclassify each input separately, while our proposed BFA attempts to misclassify the images from each object category using the identical attacked model. Consequently, the measurable success of BFA would be making the DNN generate output randomly. Therefore, we report the number of bit-flips $N_{\text{flip}}$ required to cause the DNN's test accuracy to go below 11% as the measurable indicator of BFA performance, for CIFAR-10 dataset.

As the experimental result listed in Table 24, for all the ResNet architecture with varying quantization bit-width, the required number of bit-flips $N_{\text{flip}}$ to make the DNN malfunction is most likely below 20. Besides $N_{\text{flip}}$, we take the hamming distance $D_{\text{B}}$ between clean- and perturbed-model as another measurable indicator. The intuition behind this is our proposed BFA attempts to flip the selected bits without considering its original status. Thus, it exists the probability that some of the bits might be flipped repeatedly with even times. However, the reality is that such back and forth bit-flips rarely happen throughout all the experiments. Under varying quantization configurations, there is no obvious relation between the quantization bit-width and the required number of bit-flips (i.e., the resistance against BFA).

### 5.3.3 BFA on ImageNet

The evaluation of our attack on the ImageNet dataset is presented in Table 25. We report both baseline and 8-bit quantized network accuracy for four popular image classification architectures on ImageNet. We observe roughly 0.1-0.4 % reduction in Top-1 classification accuracy after quantizing the network's weights to 8-bits. Since the ImageNet dataset has 1000 different classes of objects, a classification accuracy of 0.1% can be considered as random output. Thus reporting only the number of bit flips $N_{\text{flip}}$

Table 25: BFA on ImageNet with various network architecture, under direct 8-bit weight quantization (without retraining). Accuracy (Acc.) is in top1/top5 format. $N_{\text{flip}}$ is the median number of bit-flips (out of 5 trials) required to degrade the top-1 accuracy below 0.2%. For ImageNet, top-1 accuracy with random guess is 0.1%. $D_{\text{B}}$ is the corresponding hamming distance. Capacity is the number of bits used for weight storage (# of weights $\times$ 8).

| Model (Capacity) | Baseline Acc. % | Quantized Acc. % | $N_{\text{flip}}$ | $D_{\text{B}}$ |
|---|---|---|---|---|
| AlexNet [12] (488,806,720) | 56.55/79.08 | 56.13/78.94 | 17 | 17 |
| ResNet-18 [2] (93,516,096) | 69.76/89.08 | 69.50/88.98 | 13 | 13 |
| ResNet-34 [2] (174,381,376) | 73.30/91.42 | 73.13/91.38 | 11 | 11 |
| ResNet-50 [2] (204,456,256) | 76.15/92.87 | 75.84/92.82 | 11 | 11 |
| MobileNet-v2 [73] (27,758,080) | 71.88/90.29 | 71.14/90.01 | 1 | 1 |

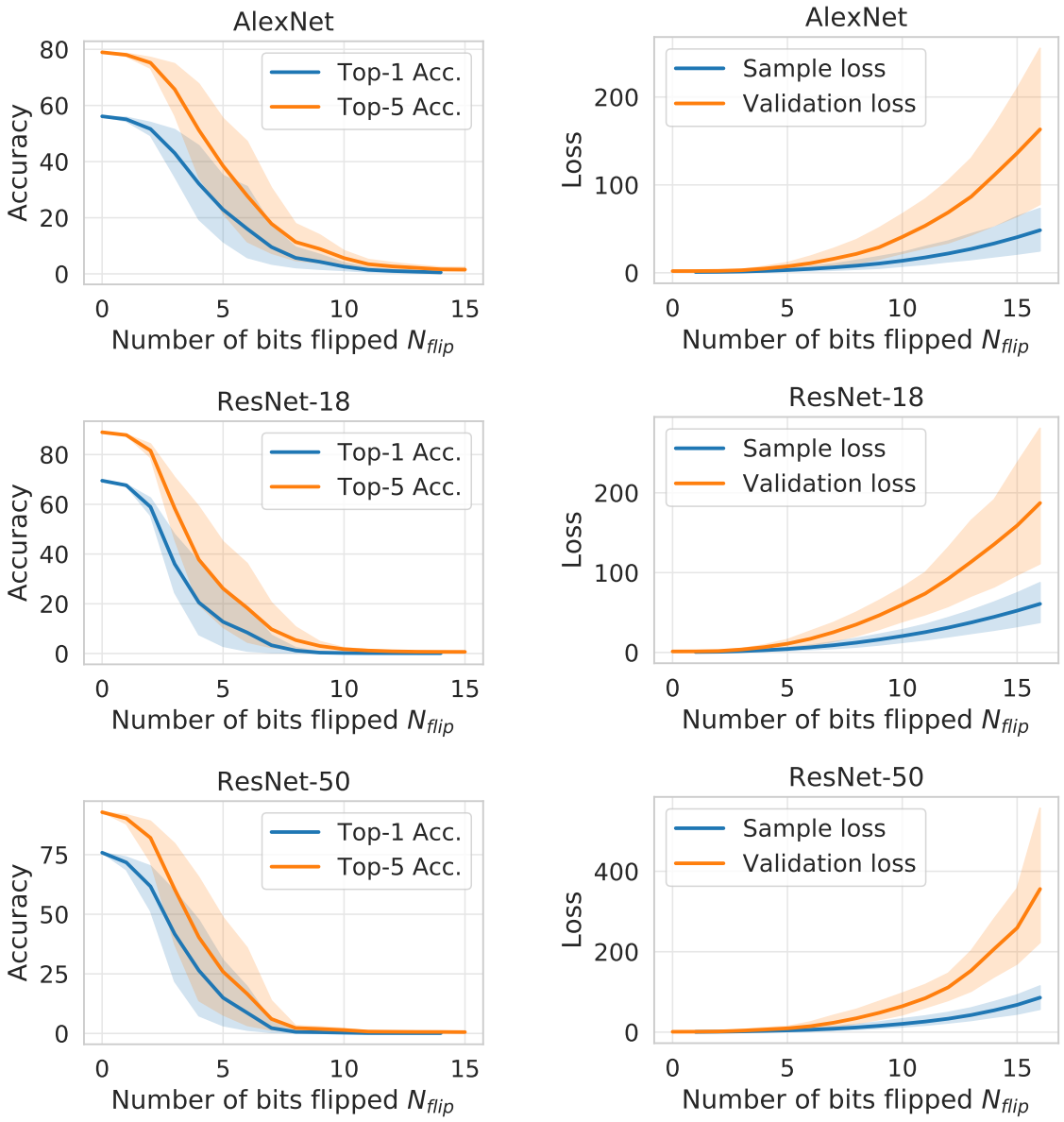



Figure 22: The accuracy (Top1/Top5) and loss evolution curve versus the number of bit-flips ($N_{\text{flip}}$) under BFA, for AlexNet/ResNet-18/ResNet-50 on ImageNet dataset. The sample size for performing BFA is 256. On each network architecture, we run 5 experiments and the region in shadow indicate the error-band. For all experiments in this figure, there exists no bit flipped multiple times during the attack (i.e., $N_{\text{flip}} = D_{\text{B}}$).

required to cause the accuracy to degrade to below 0.2% would be sufficient to prove the attack's effectiveness.

For ImageNet, BFA with PBS attack requires only 17 (median of 5 trials ) bit flips out of 480 Million bits to crush AlexNet. However, $N_{\text{flip}}$ decreases even more as we perform the attack on ResNet architectures. Fig. 22 shows accuracy degradation for ResNet models, which has a much steeper slope than AlexNet. As AlexNet does not have residual connections, which may result in a different response to such gradient-based attacks. For ResNet networks, as the network parameters keep increasing, it requires a lesser number of $N_{\text{flip}}$ to attack the network. Finally, Our attack makes a ResNet-50 architecture dysfunctional by flipping 11 out of 200 Million bits only. The attack achieves such success by modifying roughly 0.000003% of the bits to destroy the fully functional DNN. Thus the gravity of the DNN parameter's security concern can be summarized as two identical models with 50M similar weights but only a 0.000003% error in the parameters can generate completely different output values causing a 63% degradation in test accuracy.

### 5.3.4 Ablation Study

**PBS with various sample size.** In our experiment, we randomly sample a set of input images from the test/validation subset to perform the BFA, which we define as *attack sample*. Then, we evaluate the effectiveness of the attack on the whole test data set which works as a validation. We opted to perform the validation on the whole test dataset including the random batch that was originally selected for the attack because the sample size is too small compared to the whole test dataset for both ImageNet and CIFAR-10. In this section, we perform an ablation study on the attack sample size.

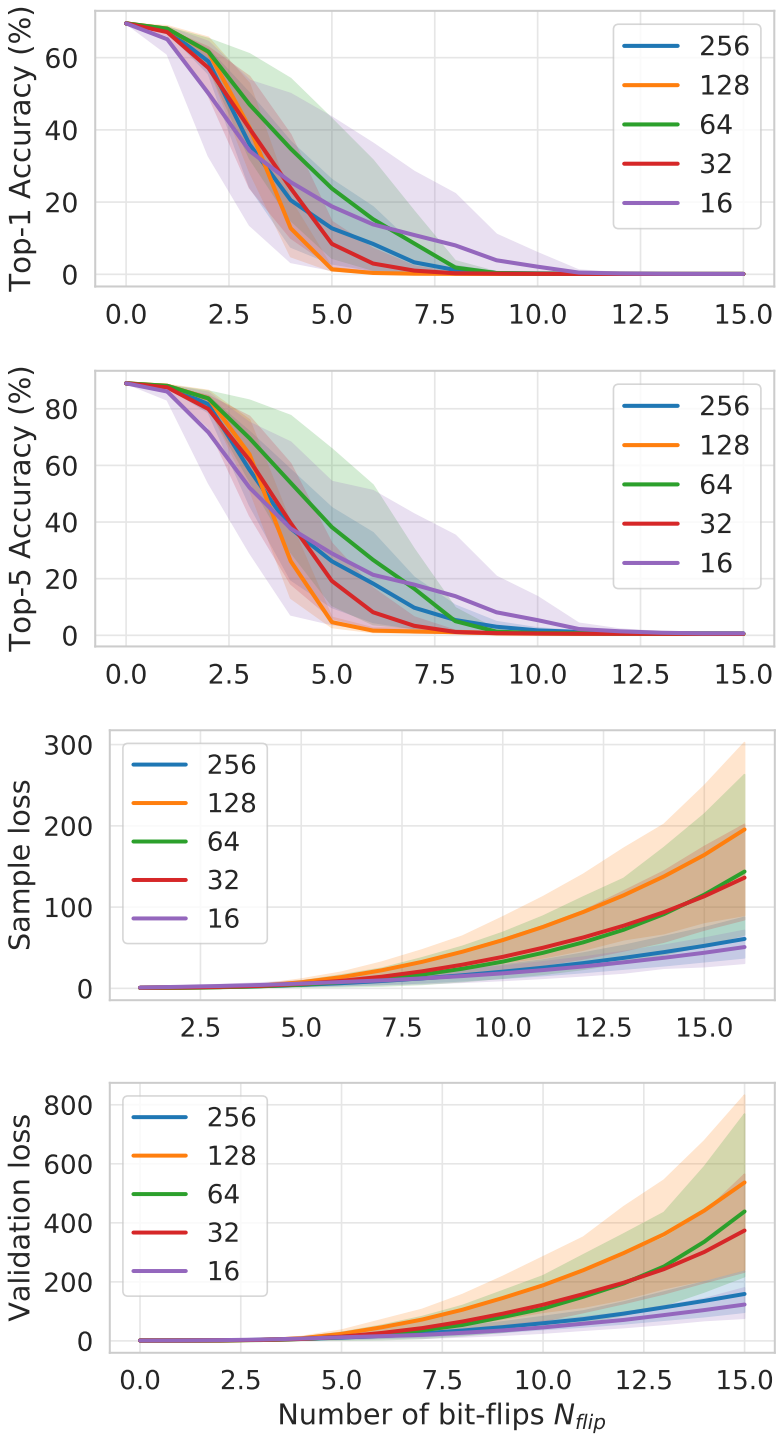



Figure 23: The BFA performance of ResNet-18 with various attack sample size (16/32/64/128/256) on ImageNet dataset. Regions in shadow indicates the error band w.r.t 5 trials.

In Fig. 23, We configure the sample size from 16-256 and plotted Top-1 validation accuracy, Top-5 validation accuracy, Sample loss and validation loss respectively.

The performance of the attack based on attack sample size can be ranked as: $S(128) > S(32) > S(256) > S(64) > S(16)$. The observation confirms that with even small input sample size, PBS still works with very small bit-flips.

**PBS versus random bit-flips.** In this section, we perform an ablation study on randomly flipping any bits of a random weight in the network. First, we test random bit-flips on a full-precision weight(i.e, floating-point) of the ResNet-18 model. For floating-point weights represented in standard IEEE format, if we change the most significant bits of the exponent section, then the floating-point weight value would change by a huge amount. As a result, the trained ResNet-18 Network starts malfunctioning even after just one random bit flip.

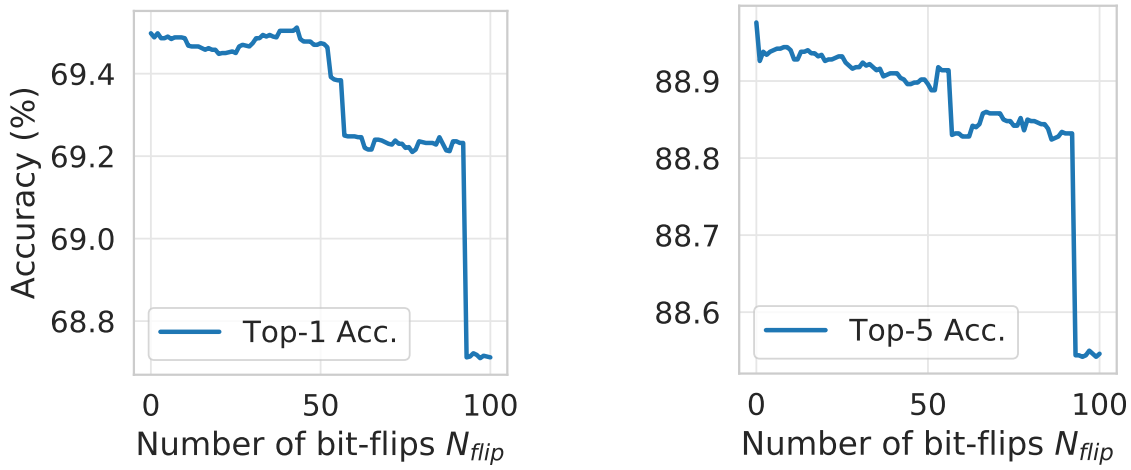



Figure 24: Randomly flipping bits of a ResNet-18 architecture on ImageNet. Even after flipping 100 random bits the network's both Top-1 and Top-5 accuracy does not degrade significantly.

Then, we implement the random bit flip on 8-bit Quantized ResNet-18 architecture as shown in Fig. 24. It shows that by flipping even 100 random bits, the Top-1 accuracy

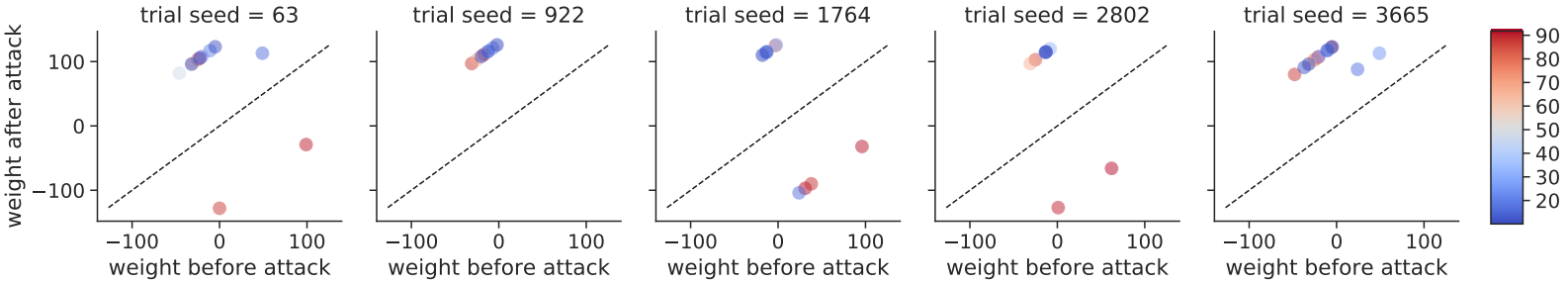
on the ImageNet dataset does not degrade more than 1%. It demonstrates the need for an efficient bit search algorithm to identify the most vulnerable bits as randomly flipping any bit does not hamper the neural network too much. In comparison, our attack algorithm requires just 13 bits out of 93M for ResNet-18 to completely cause the network to malfunction on the ImageNet dataset.

### 5.3.5 Comparison to Other Methods

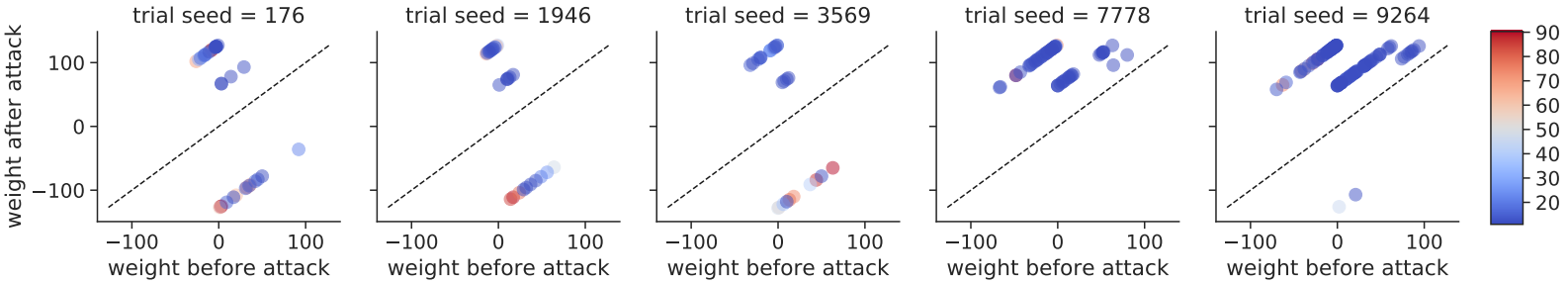
Progressive bit search is the very first attack bit searching algorithm developed to malfunction a quantized neural network through perturbation of stored model parameters using row hammer attack. We already showed in the previous section that the previous attack algorithms [51], [52] on floating-point model parameters are not efficient. They do not consider that attacking the floating-point DNN model is as easy as flipping the most significant exponent bits of any random weights. Our developed BFA with PBS is the first work that emphasizes the need for developing attack algorithms to properly scrutinize the security of DNN model parameters. Our attack can crush a DNN model to demonstrate DNN's vulnerability to intentional malicious bit flips. Further, our algorithm would encourage more future work on both attack and defense front in an attempt to make the neural networks more resilient and robust.

## 5.4 Observations of BFA

To first understand, then to defend and harness the bit-flip based adversarial weight attack, we conducted some preliminary investigations, along with several important observations as described below.




(a) ResNet-20 [2] with 8-bit weight. Bit-flips $N_{\text{BF}}$ for BFA (mean±std):$11.2 \pm 1.9$, total number of bits of weights: 2 millions.




(b) VGG-11 [14] with 8-bit weight. Bit-flips $N_{\text{BF}}$ for BFA (mean±std):$56.6 \pm 35.2$, total number of bits of weights: 78 millions.

Figure 25: Weight shift caused by BFA for (a) ResNet-20 and (b) VGG-11 on CIFAR-10 dataset. For both architectures, 5 trials are executed with different random seeds. Each colored dot depicts the weight shift (x-axis: prior-attack weight, y-axis: post-attack weight) w.r.t one iteration of BFA. The color bar indicates the corresponding accuracy (%) on the CIFAR-10 test data. The vertical distance between dot and the diagonal dashed line (i.e., $y = x$) represents the weight shift magnitude. Moreover, results reported in this figure use 8-bit post-training weight quantization.

**Observation 1** *BFA is prone to flip bits of close-to-zero weights, and cause large weight shift.*

As depicted in Fig. 25, the progressive bit search proposed in BFA is prone to identify vulnerable bit in the weight whose absolute value has a small magnitude (i.e., $|w| \rightarrow 0$) then modify it to be a large value (explained in the caption of Fig. 25). Since the BFA performs the attack on the quantized weight encoded in two's complement, the possible weight magnitude shift is discretized as $2^i, i \in \{0, 1, ..., n_{\mathrm{q}}\}$. Moreover, Fig. 25 also shows the model with larger capacity possesses higher resistance against BFA (i.e., require more bit-flips for same accuracy degradation).

**Observation 2** *BFA is prone to flip the weight bits in the front-end layers of the target neural network.*

Fig. 26 shows the histogram of bit-flips across different modules[13] of DNN under BFA. All the trials show that most of the bits found by BFA are mostly in the front-end, along the forward propagation path. Such an observation can be explained as the error introduced by the bit-flips in the front-end can be easily accumulated and amplified during the forward propagation, which is similar to the linear explanation of adversarial example discussed in [31].

**Observation 3** *BFA forces almost all the inputs to be classified into one particular output group.*

Fig. 27 depicts the top-1 categorization output of DNN on CIFAR-10 test data, at different BFA iterations. The CIFAR-10 test subset includes 1000 samples on 10 output

[13]module index includes all modules within DNN, where small to large index denotes the location from front to rear along the inference path.

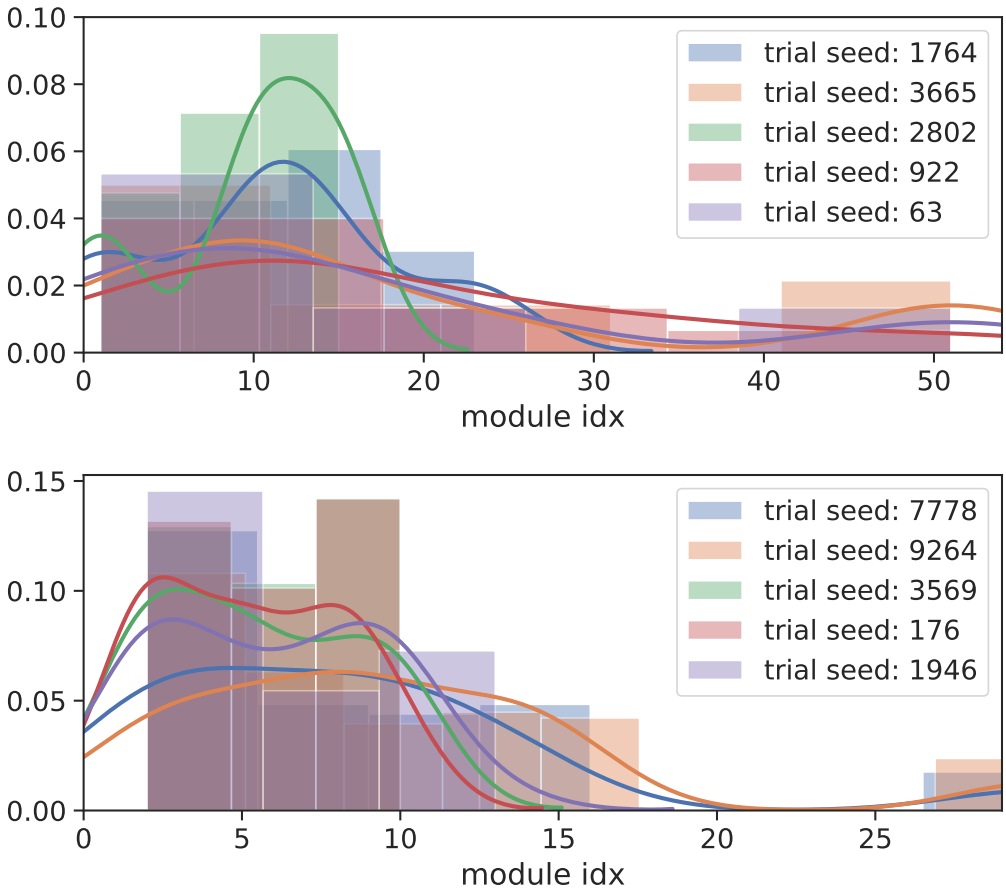



Figure 26: Normalized histogram and Kernel Density Estimation (KDE) of bit-flips versus module index in (top) ResNet-20 and (bottom) VGG-11 on CIFAR-10.

categories (i.e., total 10k samples). The BFA-free clean model, at iteration 0, has almost evenly distributed top-1 classification output predictions in each output category. It is intriguing to notice that, with the evolution of BFA, it forces almost all inputs to be classified into one output group. We also find that the dominant output group highly depends on the given attack sample data.

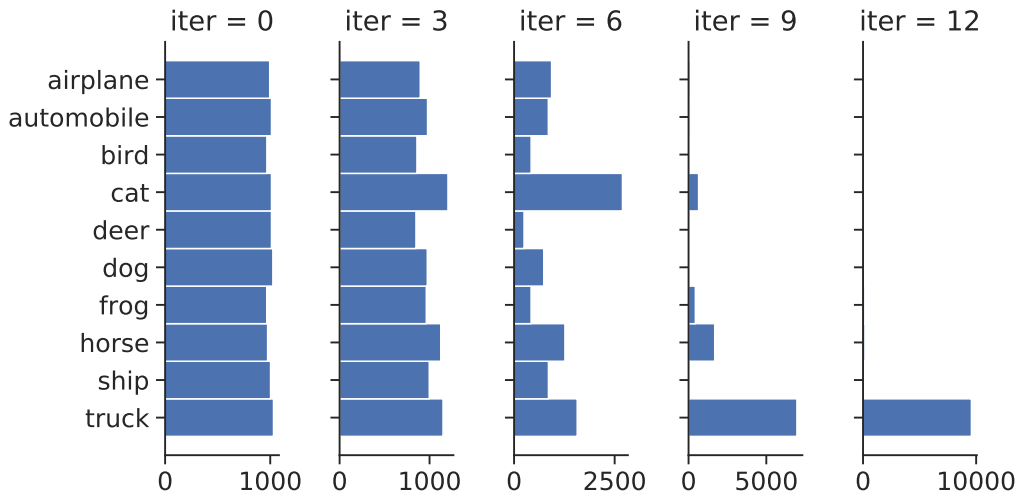



Figure 27: The evolution of ResNet-20 output classification histogram across 10 categories under BFA, on 10k test samples of CIFAR-10. The attack sample size is 128.

## 5.5 Defense against BFA

To enhance the resistance of DNN against BFA, we propose and investigate two defense techniques, i.e., Binarization-aware training and its relaxation – piece-wise weight clustering, inspired by the observations in Section 5.4.

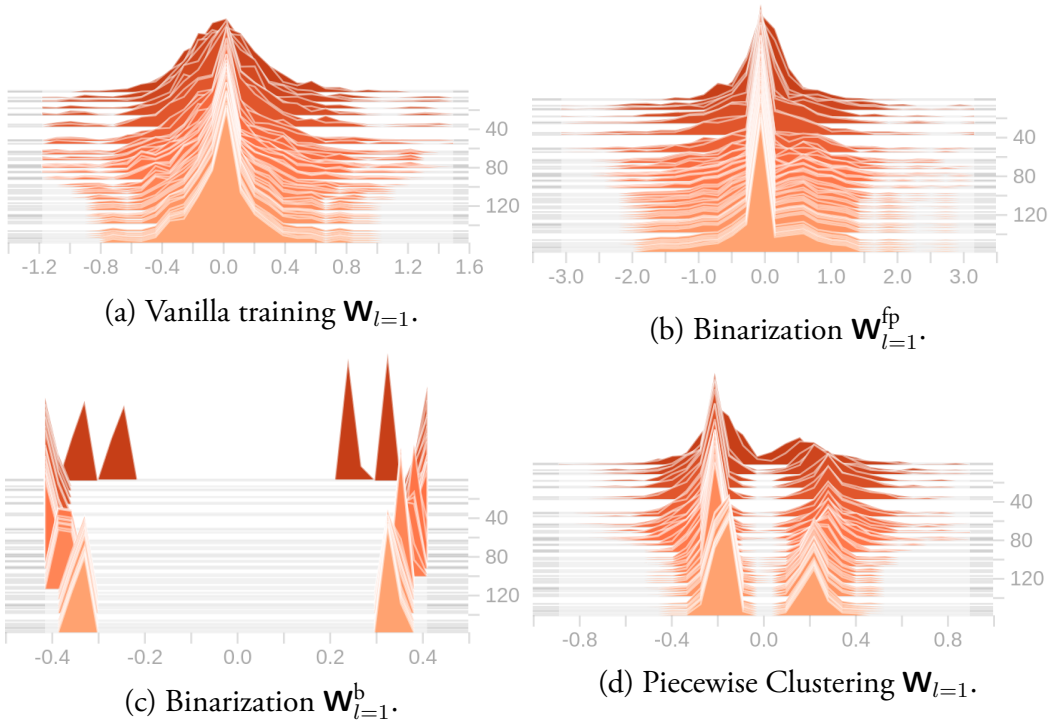
### 5.5.1 Binarization-aware Training

binarization-aware training is originally proposed as an extreme low bit-width model compression technique, which converts the weights from 32-bit floating-point to {-1,+1} binary format encoded by 1-bit [40]. Here, the binarization-aware training is leveraged as a defense technique against BFA, which can be mathematically described

as:

$$\begin{aligned} \text{Forward}: \quad & w_{l,i}^{\text{b}} = \mathbb{E}(|\mathbf{W}_l^{\text{fp}}|) \cdot \text{sgn}(w_{l,i}^{\text{fp}}) \\ \text{Backward}: \quad & \frac{\partial \mathcal{L}}{\partial w_{l,i}^{\text{b}}} = \frac{\partial \mathcal{L}}{\partial w_{l,i}^{\text{fp}}} \end{aligned} \tag{5.13}$$

where sgn() is the sign function. $w_{l,i}^{\text{b}}$ denotes the binarized weight from its floating-point counterpart $w_{l,i}^{\text{fp}}$. In general, weight binarization intrinsically achieves two goals: 1) reducing the bit-width to 1, and 2) clustering the weights to $\pm\mathbb{E}(|\mathbf{W}_l^{\text{fp}}|)$ as in Eq. (5.13). The Straight Through Estimator (STE) [83] is adopted to address the non-differential problem for the sign function as prior works [127]. Nevertheless, different from STE in [127], the gradient clipping constraint is omitted from the backward path, thanks to the presence of weight scaling coefficient $\mathbb{E}(|\mathbf{W}_l^{\text{fp}}|)$.




(a) Vanilla training $\mathbf{W}_{l=1}$.

(b) Binarization $\mathbf{W}_{l=1}^{\text{fp}}$.

(c) Binarization $\mathbf{W}_{l=1}^{\text{b}}$.

(d) Piecewise Clustering $\mathbf{W}_{l=1}$.

Figure 28: The evolution of weight distribution of ResNet-20 (first layer, $l = 1$), under various training configurations. x-axis: weight magnitude, y-axis: training epoch.

Our interpretation of the BFA resistance enhancement through binarization-aware training comes in twofold: 1) As discussed in observation-1, BFA is prone to attack the

close-to-zero weights and cause the large weight shift. The weight binarization eliminates close-to-zero weights by forcing all the weights to $\pm\mathbb{E}(|\mathbf{W}_l^{\text{fp}}|)$, where the weight distribution of sampled layer is illustrated in Fig. 28c. 2) Binarization-aware training intrinsically acts as training the DNN with bit-flip noise injected. As described in Eq. (5.13), the floating-point weight base $\{\mathbf{W}_l^{\text{fp}}\}$ are binarized on-the-fly during training. Recalling the optimization using SGD, the weight change $\Delta w^{\text{fp}}$ can be expressed as:

$$\Delta w^{\text{fp}} = \eta \cdot \nabla_{w^{\text{fp}}} \mathcal{L}(f(\boldsymbol{x}, \{\mathbf{W}^{\text{fp}}\}); \boldsymbol{t}) \tag{5.14}$$

where $\eta$ is the learning rate. Due to the presence of Eq. (5.13), even small weight update on $w^{\text{fp}}$ (i.e., $w^{\text{fp}} - \Delta w^{\text{fp}}$) may directly change the corresponding binarized weight $w^{\text{b}}$ from -1 to +1 or the opposite as a bit-flip, when the following condition is meet:

$$\text{sgn}(w^{\text{fp}} - \Delta w^{\text{fp}}) \neq \text{sgn}(w^{\text{fp}}) \tag{5.15}$$

Therefore, the binarization-aware training involves massive bit-flips on the binarized weight $\boldsymbol{W}^{\text{fp}}$, which mimics injecting the bit-flips noise on the weights during training. Fig. 29 depicts the average number of bit-flips caused by the weight update when training a binarized ResNet-20, each iteration may cause around 300 bit-flips on the binarized weights even when the learning rate is 0.001.

### 5.5.2 Clustering as Relaxation of Binarization

Since weight binarization normally suffers from significant prediction accuracy degradation due to aggressive model capacity reduction, we propose a relaxation to the weight binarization, called *Piece-wise Clustering* (PC), to emit the fixed single bit-width constraint while retaining similar functionality of clustering, which we believe

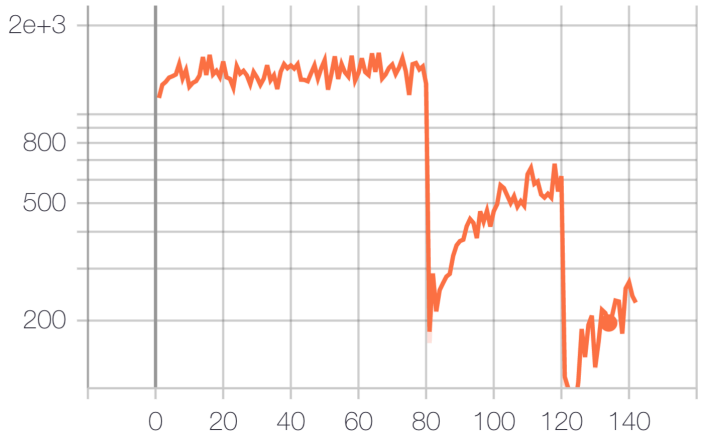



Figure 29: Average #Bit-flips (y-axis) per weight update iteration of binarization-aware training vs. epochs (x-axis), with ResNet-20 on CIFAR10.

play an important role in defending BFA. The piece-wise clustering introduces an additional weight penalty to the inference loss $\mathcal{L}$ (e.g., cross-entropy), and the optimization can be formulated as:

$$\min_{\{\mathbf{W}_l\}_{l=1}^{L}} \mathbb{E}_{\boldsymbol{x}}\, \mathcal{L}(f(\boldsymbol{x}, \{\mathbf{W}_l\}_{l=1}^{L}), \boldsymbol{t}) + \underbrace{\lambda \cdot \sum_{l=1}^{L} (||\mathbf{W}_l^{+} - \mathbb{E}(\mathbf{W}_l^{+})||_2 + ||\mathbf{W}_l^{-} - \mathbb{E}(\mathbf{W}_l^{-})||_2)}_{\text{piece-wise clustering penalty term}} \tag{5.16}$$

where $\lambda$ is the clustering coefficient to tune the strength of the weight clustering penalty term. $\mathbf{W}_l^{+}$ and $\mathbf{W}_l^{-}$ denote the positive and negative weight subset of $l$-th layer weight tensor. The DNN model optimized as Eq. (5.16) leads to a bi-modal weight distribution as depicted in Fig. 28d. The piece-wise clustering proposed above can also be viewed as a variant of group Lasso, where the group is defined as the positive and negative weight subsets in each layer.

## 5.6 Experiments of BFA Defense

### 5.6.1 Experiment Setup

**Dataset and Network Architectures** in this chapter, experiments are focused on visual dataset CIFAR-10 [126], which includes 60k $32 \times 32$ RGB images evenly sampled from 10 categories, with 50k and 10k samples for training and test respectively. The data augmentation technique is identical as reported in [2]. The ResNet-20 [2] and VGG-11 [14] are the two networks studied in the work. We use the momentum-based stochastic gradient descent optimizer, with training batch-size and weigh decay as 128 and 3e-4 respectively. The initial learning rate is 0.1 that scaled by 0.1 at 80 and 120 epochs, and the total number of epochs is 160. Note that, all the experiments are conducted using Pytorch [66], running on NVIDIA Titan-XP GPUs.

**BFA Configuration.** To evaluate the effectiveness of the proposed defense methods, the code from [53] is utilized with further modification. The number of bit-flips $N_{\text{BF}}$ that degrades the prediction accuracy below 11% is used as the metric to measure the BFA resistance, for CIFAR-10 dataset. Moreover, since BFA requires a set of data to perform the attack, we take 256 sample images from the training subset as the default BFA configuration and report the mean$\pm$std of $N_{\text{BF}}$ with 5 BFA trials. Note that, all the quantized DNN reported hereafter still uses the uniform quantizer as in [53], but with quantization-aware training instead of post-training quantization.

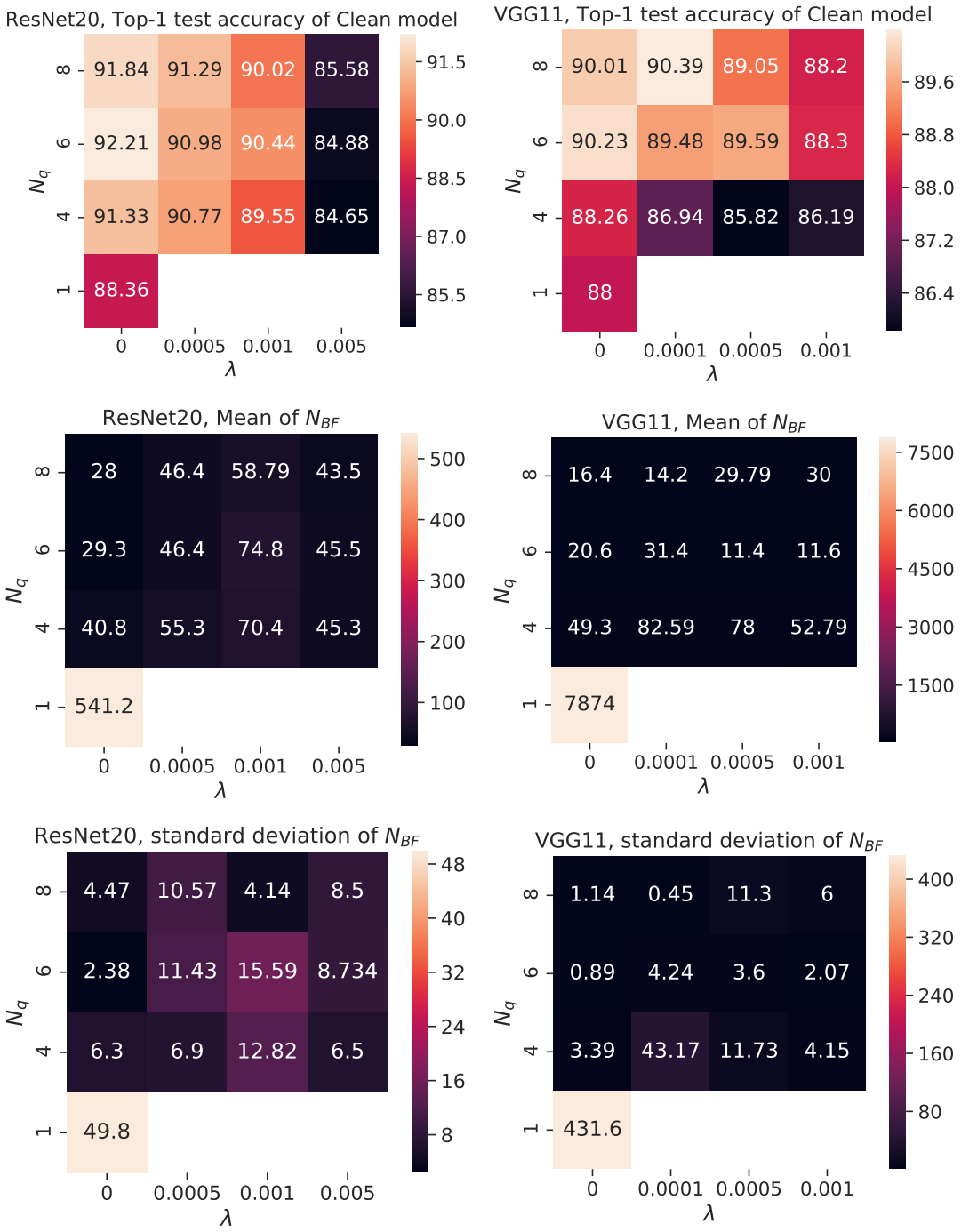



Figure 30: The BFA-free test accuracy, mean and standard deviation of $N_{\mathrm{BF}}$ for 5-trials under different quantization bit-width $N_{\mathrm{q}} \in \{8, 6, 4, 1\}$ and clustering penalty coefficient $\lambda \in \{0, 1e-4, 5e-4, 1e-3, 5e-3\}$, with (left column) ResNet-20 and (right column) VGG-11 on CIFAR-10.

### 5.6.2 Result Evaluation

The experiment results with different quantization bit-width $N_{\mathrm{q}}$ and clustering coefficient $\lambda$ of piecewise clustering are summarized in Fig. 30. It reports the BFA-free test accuracy and number of bit-flips $N_{\mathrm{BF}}$ required for BFA to succeed. Note that, for weight binarization in Fig. 30, we exclude the Piece-wise Clustering (PC) term through setting $\lambda = 0$, since binarization intrinsically performs the clustering as discussed in Section 5.5.1.

**Effect of quantization bit-width and clustering coefficient.** Based on the results reported in Fig. 30, training the DNN with binarized weights roughly degrade the test accuracy by $\sim 4\%$ and $\sim 2\%$ in comparison to the 8-bit quantized counterpart, for ResNet-20 and VGG-11. As discussed in Section 5.5.2, our intention of proposing the piece-wise clustering as the relaxation to weight binarization is to mitigate such accuracy drop. We do observe that using piece-wise clustering with proper $\lambda$ can mitigate the accuracy degradation while improving the BFA resistance (i.e., requiring more bit-flips $N_{\mathrm{BF}}$ for the same accuracy degradation). For ResNet-20 and VGG-11, the ideal configurations of $\lambda$ are 0.001 and 0.0005 respectively, as the model with larger capacity benefits from relatively smaller $\lambda$.

**BFA resistance of ResNet-20.** The 8-bit quantized ResNet-20 (baseline) requires only an average of 28 bit-flips to hamper the functionality of an accurate DNN, while the weight binarization significantly improves the BFA resistance compared to the baseline. Binarization increases the average value of $N_{\mathrm{BF}}$ to 541.2, which improve the BFA resistance by $\sim 19\times$. Nevertheless, considering the inevitable accuracy drop due to the drastic bit-width reduction (32-bit to 1-bit), as an alternative approach we explore the performance of PC on other bit-width configurations as well. With $\lambda = 1e-3$, the

average value of $N_{\text{BF}}$ was improved by $2.09\times$, $2.55\times$ and $1.73\times$ for 8, 6 and 4 bit-width respectively. In conclusion, our proposed piece-wise clustering improves the resistance to adversarial weight attack for all the cases of different bit-widths. Still, the binarized network emerges as the most successful defense against BFA.

**BFA resistance of VGG-11.** For VGG-11, our observation follows a similar pattern as described in the previous section. The baseline VGG-11 (e.g., $N_{\text{q}} = 8$) requires an average $N_{\text{BF}}$ of 16.4. Again, weight binarization improves the network robustness significantly, yielding an average $N_{\text{BF}}$ of 7874; which is $\sim 480\times$ improvement in comparison to the baseline. In the case of VGG-11, the lower Bit-Width defends BFA even better than ResNet-20; the main reason for this discrepancy can be the difference between the size of the network. For a larger network such as VGG-11, low Bit-Width and PC performs a proper regularization to successfully defend against BFA. The best performance of PC for VGG-11 was achieved for a 4-bit network with $\lambda = 1e-4$ achieving an average value of $N_{\text{BF}}$ of 82.59.

In summary, both the binarization-aware training and its piecewise clustering relaxation can improve the BFA resistance of the target neural network, while the binarization-aware training can push the $N_{\text{BF}}$ to an extremely large value (e.g., $N_{\text{BF}} > 7000$ on over-parameterized VGG-11). The implication of such large value is noteworthy, as a larger value of $N_{\text{BF}}$ indicates the significant increase in difficulty to carry out a memory fault injection through the feasible cyber-physical attacks. For example, when using the row-hammer attack to perform the fault injection, the increased attack execution time might be detected by the operating system through the data integrity check.

### 5.6.3 Comparison of Alternative Defense Methods

Adversarial weight attack [53] is a recently developed security threat model for modern DNN. Subsequently, the development of defensive approaches in this field has not received much attention. Therefore, for the first time, we investigate an alarming parameter security concern – bit-flip based adversarial weight attack, with corresponding defense method. Owing to the lack of competing methods in this research direction, we attempt to transfer several conventional defense methods of adversarial examples [33], [128], [129], for providing a comprehensive comparison.

Table 26: Alternative Methods Comparison. In this table, we report the prior- and post-attack test accuracy (%) and $N_{\text{BF}}$ of BFA. The 8-bit quantization is chosen as the baseline; Binary and PC-8bit is the proposed method. Moreover, comparison with Lasso-based pruning and adversarial weight training (adv. training) is included as well.

| Methods | Prior-Attack Accuracy (%) | Post-Attack Accuracy (%) | $N_{\text{BF}}$ |
|---|---|---|---|
| 8-bit | 91.84 | 10.45 | 28.0±4.47 |
| PC-8bit | 90.02 | 10.07 | **58.79±4.14** |
| Binary | 88.36 | 10.13 | **541.2 ± 49.8** |
| Lasso Pruning | 88.11 | 10.12 | 6.8±0.44 |
| Adv. Training | 87.72 | 10.09 | 9.6±6.58 |

**Weight Pruning.** Both the activation and weight pruning have been investigated as the defense against adversarial example [128], [130]. Such pruning techniques involve the stochastic process during the inference which suffers from gradient obfuscation [108] which is a common reason for the failure of adversarial input defenses. Nevertheless, we investigate the effectiveness of network pruning to resist adversarial weight attack as an alternative approach. To achieve this, we train a regular network with Lasso loss function to shrink most of the weights to an extremely low value. Thus,

we can rewrite the loss function with additional $L_1$-norm penalty as:

$$\min_{\{\mathbf{W}\}} \mathcal{L}(f(\boldsymbol{x}, \{\mathbf{W}_l\}); \boldsymbol{t}) + \beta \cdot \sum_{l=1}^{L} ||\mathbf{W}_l||_1 \tag{5.17}$$

where $\beta$ is the coefficient to tune the pruning strength. Through training with Eq. (5.17), we expect the weight tenor to be in a highly sparse representation.

The intuition behind pruning working as an adversarial weight defense can be summarized as: In a sparse network, we consider these zero-valued weights will not have any physical connection (pruned) to conduct a bit-flip attack, thus making them immune from BFA. As a result, the attacker is left with only a few portions of the weights which he/she can alter to perform the BFA. Nevertheless, as shown in Table 26, such a sparse regularized network is even more vulnerable to adversarial weight attack, requiring on average just 6.8 bit-flips to hamper the functionality of target DNN. Since a large portion of the weights was pruned, the remaining weights contain large significance in maintaining accurate network performance. So altering any of the remaining non-zero weights still manages to degrade network performance significantly.

**Adversarial Weight Training.** Inspired by the adversarial training [31], [33], we attempt to adopt the same idea and create a BFA-based adversarial training as an alternative approach to compare with our proposed method. We modified the adversarial training objective for adversarial example defense to serve the purpose of adversarial weight training:

$$\begin{gathered} \min_{\{\mathbf{B}\}} \mathcal{L}(f(\boldsymbol{x}; \{\mathbf{B}\}), \boldsymbol{t}) + \alpha \mathcal{L}(f(\boldsymbol{x}; \{\mathbf{B}\} + \{\mathbf{B}_{\text{BFA}}\}), t) \\ s.t.\ \{\mathbf{B}_{\text{BFA}}\} = \{\hat{\mathbf{B}}\} - \{\mathbf{B}\} \end{gathered} \tag{5.18}$$

where $\{\mathbf{B}_{\text{BFA}}\}$ is the different between BFA-perturbed weight bits $\{\hat{\mathbf{B}}\}$ and its BFA-free counterpart $\{\mathbf{B}\}$. During the model training, $\{\mathbf{B}_{\text{BFA}}\}$ is run-time generated as the additive constant offset on the model weights.

The result of adversarial training is shown in Table 26, where it does not show the improvement of BFA resistance from such adversarial weight training. Our interpretation of the defense failure is summarized in the following. When performing the adversarial training with the adversarial examples, each natural image owns similar adversarial examples even with a different random seed. However, for the bit-flip based adversarial weight attack, using a single natural image as the attack sample will lead to massive different combinations of vulnerable weight bits, while BFA just provides one combination in a greedy way. Thus, performing the adversarial weight training with all the vulnerable weight bit combinations is not a feasible approach. In the end, through the comparison of all the potential defense methods listed in Table 26, we conclude that the binarization-aware training and the piecewise clustering are the effective defense methods.

## 5.7 Summary

Our proposed attack is the very first work for vulnerable bit search on quantized neural networks. BFA puts light on why the security analysis for neural network parameters needs more attention. We demonstrate through extensive experiments and analysis that the vulnerability of the DNN parameter to malicious bit-flips is extremely severe than anticipated. We would encourage further investigation on both attack and defense front to thrive towards developing a more resilient network for deep learning applications. Besides, we try to develop a comprehensive investigation of adversarial parameter security enhancement and provide several insightful observations for the future struggle against DNN parameter vulnerability. Based on those observations, we highlight potential successful directions to develop adversarial weight defense. Finally, through

the comprehensive experiments, our proposed methods, especially binarization-aware training, are proven to improve the resistance against the emerging bit-flip based adversarial weight attack.

# Chapter 6

# CONCLUSIONS AND OUTLOOK

In this thesis, we have discussed my investigations on the topic of efficient and secure neural network deep learning inference systems. In terms of efficiency, leveraging the analog crossbar to perform the in-memory computing for DNN acceleration is a promising direction. However, the various non-ideal effects cause the device-to-device variation, still need extra efforts to overcome. Current crossbar accelerator simulation involves behavior modeling of circuit and device level, which can easily consume all the available memory on a 4-way GPU high-performance computing system. It still requires more hardware-efficient modeling techniques to realize the accurate simulation of large-scale neural networks and datasets. The hardware non-ideal effect is mathematically modeled and treated as the noise to perform noise-injection training, thus making the deployed neural network can tolerate those hardware non-ideal effects.

In terms of security, a noise injection variant is proposed to improve the DNN resistance against the famous adversarial example, which indicates a proper amount of the noise can helping to regularize the target DNN. Then, we extend the adversarial attack concept to a barely investigated paradigm – adversarial weight attack, where a few malicious bit-flips can fully destroy the functionality of the target DNN. Amazingly, the most effective defense method we found is weight binarization, which can also be viewed as a noise injection (i.e., bit-flip noise and quantization noise) training. all the discussions above reveal the possibility that the DNN efficiency and security could be achieved simultaneously.

The future research in this direction could be: 1) Automatic DNN generation to produce high-accuracy binary neural network, thus compensating the accuracy degradation result from the aggressive low-bit quantization (i.e., binarization); 2) The general-purpose noise injection method to improve the DNN generalization and its corresponding theory. We believe this thesis will stimulate the research effort to develop a deep learning inference system with higher efficiency and security performance.

# REFERENCES


[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

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

[3] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[4] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*, Springer, 2016, pp. 21–37.

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[6] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[7] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[8] T. Gebru, J. Krause, Y. Wang, D. Chen, J. Deng, E. L. Aiden, and L. Fei-Fei, "Using deep learning and google street view to estimate the demographic makeup of neighborhoods across the united states," *Proceedings of the National Academy of Sciences*, vol. 114, no. 50, pp. 13 108–13 113, 2017.

[9] 2018. [Online]. Available: https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/.

[10] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[11] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[13] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *ACM SIGARCH computer architecture news*, vol. 23, no. 1, pp. 20–24, 1995.

[14] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[15] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[16] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[17] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.

[18] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1492–1500.

[19] M. Horowitz, "Energy table for 45nm process," *Stanford VLSI wiki*,

[20] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *International Conference on Learning Representations (ICLR)*, 2016.

[21] Z. He, B. Gong, and D. Fan, "Optimize deep convolutional neural network with ternarized weights and high accuracy," in *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, IEEE, 2019, pp. 913–921.

[22] Z. He and D. Fan, "Simultaneously optimizing weight and quantizer of ternary neural network using truncated gaussian approximation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 438–11 446.

[23] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in neural information processing systems*, 2016, pp. 2074–2082.

[24] L. Yang, Z. He, and D. Fan, "Harmonious coexistence of structured weight pruning and ternarization for deep neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

[25] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze, *et al.*, "Tvm: An automated end-to-end optimizing compiler for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 578–594.

[26] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[27] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping cnn onto embedded fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 35–47, 2018.

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

[29] L. Yang, Z. He, and D. Fan, "Binarized depthwise separable neural network for object tracking in fpga," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, 2019, pp. 347–350.

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

[31] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *ICLR*, 2015.

[32] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[33] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *International Conference on*

*Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=rJzIBfZAb.

[34] Z. He, A. S. Rakin, and D. Fan, "Parametric noise injection: Trainable randomness to improve deep neural network robustness against adversarial attack," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 588–597.

[35] K. Xu, G. Zhang, S. Liu, Q. Fan, M. Sun, H. Chen, P.-Y. Chen, Y. Wang, and X. Lin, "Adversarial t-shirt! evading person detectors in a physical world," *arXiv*, arXiv–1910, 2019.

[36] S. Thys, W. Van Ranst, and T. Goedemé, "Fooling automated surveillance cameras: Adversarial patches to attack person detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2019.

[37] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust physical-world attacks on deep learning visual classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1625–1634.

[38] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in neural information processing systems*, 2015, pp. 3123–3131.

[39] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[40] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*, Springer, 2016, pp. 525–542.

[41] C. Leng, Z. Dou, H. Li, S. Zhu, and R. Jin, "Extremely low bit neural network: Squeeze the last bit out with admm," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[42] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, J. J. Yang, and R. S. Williams, "Dot-product engine for neuromorphic computing: Programming 1t1m crossbar to accelerate matrix-vector multiplication," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, IEEE, 2016, pp. 1–6.

[43] Z. He, J. Lin, R. Ewetz, J.-S. Yuan, and D. Fan, "Noise injection adaption: End-to-end reram crossbar non-ideal effect adaption for neural network mapping," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[44] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh, "Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, ACM, 2017, pp. 15–26.

[45] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Security and Privacy (SP), 2017 IEEE Symposium on*, IEEE, 2017, pp. 39–57.

[46] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into transferable adversarial examples and black-box attacks," *arXiv preprint arXiv:1611.02770*, 2016.

[47] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ACM, 2017, pp. 506–519.

[48] X. Liu, Y. Li, C. Wu, and C.-J. Hsieh, "Adv-bnn: Improved adversarial defense through robust bayesian neural network," *International Conference on Learning Representations (ICLR)*, 2019.

[49] M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana, "Certified robustness to adversarial examples with differential privacy," in *2019 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2019, pp. 656–672.

[50] X. Liu, M. Cheng, H. Zhang, and C.-J. Hsieh, "Towards robust neural networks via random self-ensemble," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 369–385.

[51] Y. Liu, L. Wei, B. Luo, and Q. Xu, "Fault injection attack on deep neural network," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, IEEE, 2017, pp. 131–138.

[52] J. Breier, X. Hou, D. Jap, L. Ma, S. Bhasin, and Y. Liu, "Deeplaser: Practical fault attack on deep neural networks," *arXiv preprint arXiv:1806.05859*, 2018.

[53] A. S. Rakin, Z. He, and D. Fan, "Bit-flip attack: Crushing neural network with progressive bit search," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 1211–1220.

[54] C. Zhu *et al.*, "Trained ternary quantization," *International Conference on Learning Representations (ICLR)*, 2016.

[55] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv preprint arXiv:1605.04711*, 2016.

[56] C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu, and Y. Xie, "Overcoming the challenges of crossbar resistive memory architectures," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, 2015, pp. 476–488.

[57] L. Xia, W. Huangfu, T. Tang, X. Yin, K. Chakrabarty, Y. Xie, Y. Wang, and H. Yang, "Stuck-at fault tolerance in rram computing systems," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 1, pp. 102–115, 2018.

[58] B. Liu, H. Li, Y. Chen, X. Li, T. Huang, Q. Wu, and M. Barnell, "Reduction and ir-drop compensations techniques for reliable neuromorphic computing systems," in *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '14, San Jose, California: IEEE Press, 2014, pp. 63–70. [Online]. Available: http://dl.acm.org/citation.cfm?id=2691365.2691377.

[59] L. Kish and C. Granqvist, "Noise in nanotechnology," *Microelectronics Reliability*, vol. 40, no. 11, pp. 1833–1837, 2000.

[60] D. Ielmini, F. Nardi, and C. Cagli, "Resistance-dependent amplitude of random telegraph-signal noise in resistive switching memories," *Applied Physics Letters*, vol. 96, no. 5, p. 053 503, 2010.

[61] F. M. Puglisi, L. Larcher, A. Padovani, and P. Pavan, "A complete statistical investigation of rtn in $HfO_2$-based rram in high resistive state," *IEEE Transactions on Electron Devices*, vol. 62, no. 8, pp. 2606–2613, 2015.

[62] I. Chakraborty, D. Roy, and K. Roy, "Technology aware training in memristive neuromorphic systems based on non-ideal synaptic crossbars," *arXiv preprint arXiv:1711.08889*, 2017.

[63] C. Liu, M. Hu, J. P. Strachan, and H. Li, "Rescuing memristor-based neuromorphic design with high defects," in *Design Automation Conference (DAC), 2017 54th ACM/EDAC/IEEE*, IEEE, 2017, pp. 1–6.

[64] L. Chen, J. Li, Y. Chen, Q. Deng, J. Shen, X. Liang, and L. Jiang, "Accelerator-friendly neural-network training: Learning variations and defects in rram crossbar," in *Proceedings of the Conference on Design, Automation & Test in Europe*, European Design and Automation Association, 2017, pp. 19–24.

[65] S. Jain, A. Sengupta, K. Roy, and A. Raghunathan, "Rx-caffe: Framework for evaluating and training deep neural networks on resistive crossbars," *arXiv preprint arXiv:1809.00072*, 2018.

[66] P. Adam, G. Sam, C. Soumith, C. Gregory, Y. Edward, D. Zachary, L. Zeming, D. Alban, A. Luca, and L. Adam, "Automatic differentiation in pytorch," in *Proceedings of Neural Information Processing Systems*, 2017.

[67] Z. He, A. S. Rakin, J. Li, C. Chakrabarti, and D. Fan, "Defending and harnessing the bit-flip based adversarial weight attack," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[68] Z. He, L. Yang, S. Angizi, A. S. Rakin, and D. Fan, "Sparse bd-net: A multiplication-less dnn with sparse binarized depth-wise separable convolution," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 16, no. 2, pp. 1–24, 2020.

[69] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[70] X. Ma, S. Lin, S. Ye, Z. He, L. Zhang, G. Yuan, S. Huat Tan, Z. Li, D. Fan, X. Qian, *et al.*, "Non-structured dnn weight pruning–is it beneficial in any platform?" *arXiv*, arXiv–1907, 2019.

[71] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[72] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[73] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[74] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6848–6856.

[75] S. Angizi, Z. He, A. Awad, and D. Fan, "Mrima: An mram-based in-memory accelerator," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 5, pp. 1123–1136, 2019.

[76] S. Angizi, Z. He, and D. Fan, "Parapim: A parallel processing-in-memory accelerator for binary-weight deep neural networks," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, 2019, pp. 127–132.

[77] S. Angizi, Z. He, and D. Fan, "Dima: A depthwise cnn in-memory accelerator," in *Proceedings of the International Conference on Computer-Aided Design*, ACM, 2018, p. 122.

[78] Z. He, S. Angizi, and D. Fan, "Accelerating low bit-width deep convolution neural network in mram," in *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, IEEE, 2018, pp. 533–538.

[79] S. Angizi, Z. He, A. S. Rakin, and D. Fan, "Cmp-pim: An energy-efficient comparator-based processing-in-memory neural network accelerator," in *Proceedings of the 55th Annual Design Automation Conference*, 2018, pp. 1–6.

[80] J. G. Proakis, M. Salehi, N. Zhou, and X. Li, *Communication systems engineering*. Prentice Hall New Jersey, 1994, vol. 2.

[81] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[82] Z.-G. Liu and M. Mattina, "Learning low-precision neural networks without straight-through estimator (ste)," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, AAAI Press, 2019, pp. 3066–3072.

[83] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[84] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[85] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, "Pact: Parameterized clipping activation for quantized neural networks," *arXiv preprint arXiv:1805.06085*, 2018.

[86] J. Choi, S. Venkataramani, V. Srinivasan, K. Gopalakrishnan, Z. Wang, and P. Chuang, "Accurate and efficient 2-bit quantized neural networks," in *Proceedings of the 2nd SysML Conference*, 2019.

[87] S. Migacz, "8-bit inference with tensorrt," in *GPU technology conference*, vol. 2, 2017, p. 5.

[88] X. Lin, C. Zhao, and W. Pan, "Towards accurate binary convolutional neural network," in *Advances in Neural Information Processing Systems*, 2017, pp. 344–352.

[89] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," in *Solid-State Circuits Conference (ISSCC), 2016 IEEE International*, IEEE, 2016, pp. 262–263.

[90] W. Tang, G. Hua, and L. Wang, "How to train a compact binary neural network with high accuracy?" In *AAAI*, 2017, pp. 2625–2631.

[91] A. Ehliar, "Area efficient floating-point adder and multiplier with ieee-754 compatible semantics," in *Field-Programmable Technology (FPT), 2014 International Conference on*, IEEE, 2014, pp. 131–138.

[92] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[93] Z. He and D. Fan, "A tunable magnetic skyrmion neuron cluster for energy efficient artificial neural network," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, IEEE, 2017, pp. 350–355.

[94] S. Angizi, Z. He, D. Reis, X. S. Hu, W. Tsai, S. J. Lin, and D. Fan, "Accelerating deep neural networks in processing-in-memory platforms: Analog or digital approach?" In *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, IEEE, 2019, pp. 197–202.

[95] K. Ueyoshi, K. Ando, K. Hirose, S. Takamaeda-Yamazaki, J. Kadomoto, T. Miyata, M. Hamada, T. Kuroda, and M. Motomura, "Quest: A 7.49 tops multi-purpose log-quantized dnn inference engine stacked on 96mb 3d sram using inductive-coupling technology in 40nm cmos," in *Solid-State Circuits Conference-(ISSCC), 2018 IEEE International*, IEEE, 2018, pp. 216–218.

[96] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "Drisa: A dram-based reconfigurable in-situ accelerator," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ACM, 2017, pp. 288–301.

[97] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ACM SIGARCH Computer Architecture News*, IEEE Press, vol. 44, 2016, pp. 27–39.

[98] C. Xu, D. Niu, N. Muralimanohar, N. P. Jouppi, and Y. Xie, "Understanding the trade-offs in multi-level cell reram memory design," in *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*, IEEE, 2013, pp. 1–6.

[99] C.-Y. Chen, H.-C. Shih, C.-W. Wu, C.-H. Lin, P.-F. Chiu, S.-S. Sheu, and F. T. Chen, "Rram defect modeling and failure analysis based on march test and a novel squeeze-search scheme," *IEEE Transactions on Computers*, vol. 64, no. 1, pp. 180–190, 2015.

[100] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*, IEEE, 2017, pp. 541–552.

[101] S. Yu, P. Chen, Y. Cao, L. Xia, Y. Wang, and H. Wu, "Scaling-up resistive synaptic arrays for neuro-inspired architecture: Challenges and prospect," in *2015 IEEE International Electron Devices Meeting (IEDM)*, 2015, pp. 17.3.1–17.3.4.

[102] F. Alibart, L. Gao, B. D. Hoskins, and D. B. Strukov, "High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm," *Nanotechnology*, vol. 23, no. 7, p. 075 201, 2012.

[103] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

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

[105] L. T. Pillage, R. A. Rohrer, and C. Visweswariah, *Electronic circuit and system simulation methods*. McGraw-Hill New York, 1995.

[106] J. Buckman, A. Roy, C. Raffel, and I. Goodfellow, "Thermometer encoding: One hot way to resist adversarial examples," *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=S18Su--CW.

[107] P. Samangouei, M. Kabkab, and R. Chellappa, "Defense-GAN: Protecting classifiers against adversarial attacks using generative models," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=BkJ3ibb0-.

[108] A. Athalye, N. Carlini, and D. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," *arXiv preprint arXiv:1802.00420*, 2018.

[109] A. Bietti, G. Mialon, and J. Mairal, "On regularization and robustness of deep neural networks," *arXiv preprint arXiv:1810.00363*, 2018.

[110] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *International Conference on Machine Learning*, 2013, pp. 1058–1066.

[111] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html.

[112] G. K. Santhanam and P. Grnarova, "Defending against adversarial attacks by leveraging an entire gan," *arXiv preprint arXiv:1805.10652*, 2018.

[113] C. Xie, J. Wang, Z. Zhang, Z. Ren, and A. Yuille, "Mitigating adversarial effects through randomization," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=Sk9yuql0Z.

[114] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[115] H. Qian and M. N. Wegman, "L2-nonexpansive neural networks," *arXiv preprint arXiv:1802.07896*, 2018.

[116] A. Athalye and N. Carlini, "On the robustness of the cvpr 2018 white-box adversarial example defenses," *arXiv preprint arXiv:1804.03286*, 2018.

[117] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," in *ACM SIGARCH Computer Architecture News*, IEEE Press, vol. 42, 2014, pp. 361–372.

[118] K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida, and H. Bos, "Flip feng shui: Hammering a needle in the software stack," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 1–18.

[119] L. Cojocar, K. Razavi, C. Giuffrida, and H. Bos, "Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks," in *2019 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2019, pp. 55–71.

[120] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoechl, and Y. Yarom, "Another flip in the wall of rowhammer defenses," in *2018 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2018, pp. 245–261.

[121] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE transactions on neural networks and learning systems*, 2019.

[122] J. Clements and Y. Lao, "Hardware trojan attacks on neural networks," *arXiv preprint arXiv:1806.05768*, 2018.

[123] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," in *Network and Distributed Systems Security (NDSS) Symposium 2018*, 2018.

[124] "Ieee standard for floating-point arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, 2019.

[125] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[126] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)," *URL http://www. cs. toronto. edu/kriz/cifar. html*, 2010.

[127] I. Hubara *et al.*, "Binarized neural networks," in *Advances in neural information processing systems*, 2016, pp. 4107–4115.

[128] G. S. Dhillon, K. Azizzadenesheli, Z. C. Lipton, J. Bernstein, J. Kossaifi, A. Khanna, and A. Anandkumar, "Stochastic activation pruning for robust adversarial defense," *arXiv preprint arXiv:1803.01442*, 2018.

[129] S. Wang, X. Wang, S. Ye, P. Zhao, and X. Lin, "Defending dnn adversarial attacks with pruning and logits augmentation," in *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, IEEE, 2018, pp. 1144–1148.

[130] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019, pp. 8612–8620.

# BIOGRAPHICAL SKETCH

Zhezhi He has been pursuing a Ph.D. degree in the School of Electrical, Computer, and Energy Engineering at Arizona State University, under the supervision of Dr. Deliang Fan. Before that, he received the M.Eng. degree in electrical and computer engineering from Oregon State University, Corvallis, OR, USA, in 2015, and the B.S. degree in Information Engineering from Southeast University, Nanjing, China, in 2012. His research interests are in the area of secure and efficient deep learning, brain-inspired in-memory computing, and post-CMOS technologies. During his past four and a half years of Ph.D. study, he has been the primary author of more than 50 publications on IEEE/ACM top-tier journal and conference (e.g., CVPR, ICCV, AAAI, DAC, DATE, ICCAD, ISLPED, TCAD, TETC, etc) in the aforementioned areas. His works also result in the receipt of multiple fellowships, scholarships, and 2 best paper awards from IEEE ISVLSI 2017 and 2018, respectively. He is also serving as the technical program committee and reviewer for various conferences (e.g., NeurIPS, CVPR, HPCA, MICRO, DAC, DATE, ICCAD, etc.) and Journals (TNNLS, TPDS, AI, TCAS-I, TED, JETCAS, etc.).