Bit-Flip Attack: Crushing Neural Network with Progressive Bit Search

Adnan Siraj Rakin[†], Zhezhi He[†] and Deliang Fan
Dept. of Electrical and Computer Engineering, Arizona State University, Tempe, AZ 85287
[†] These authors contributed equally
{asrakin, Zhezhihe, dfan}@asu.edu

Abstract

Several important security issues of Deep Neural Network (DNN) have been raised recently associated with different applications and components. The most widely investigated security concern of DNN is from its malicious input, a.k.a adversarial example. Nevertheless, the security challenge of DNN's parameters is not well explored yet. In this work, we are the first to propose a novel DNN weight attack methodology called Bit-Flip Attack (BFA) which can crush a neural network through maliciously flipping extremely small amount of bits within its weight storage memory system (i.e., DRAM). The bit-flip operations could be conducted through well-known Row-Hammer attack, while our main contribution is to develop an algorithm to identify the most vulnerable bits of DNN weight parameters (stored in memory as binary bits), that could maximize the accuracy degradation with a minimum number of bit-flips. Our proposed BFA utilizes a Progressive Bit Search (PBS) method which combines gradient ranking and progressive search to identify the most vulnerable bit to be flipped. With the aid of PBS, we can successfully attack a ResNet-18 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%. Code is released at:* `https://github.com/elliothe/Neural_Network_Weight_Attack`

1. Introduction

Recently, deep neural networks (DNNs) have demonstrated its great potential of surpassing or close to human-level performance in multiple domains, such as object recognition [14], Game AI [34], synthetic voice [27], neighborhood voting prediction [10] and etc [9]. It stimulates the demand for deploying state-of-the-art deep learning algorithms in real-world applications to release labors from repetitive work. Under such circumstance, the security and robustness of deep neural network is an essential concern which cannot be circumvented.

Adversarial example [11] (aka., adversarial attack) is a well-known security issue of DNN, which can cause the system malfunction with the magnitude-constrained input noise that mankind cannot discern. Both attack and defense of adversarial example on the input end of DNN has been heavily investigated in the past couple of years [26, 11, 36] and still be in progress [16, 29, 22]. Nevertheless, the security issue of network parameters themselves is not yet well explored. Recently, the development of fault injection attack [25] has raised further security concerns on the storage of DNN parameters.

The possible reasons that there was a lack of concerns on the security of network parameters may come in twofold: 1) The neural network is widely recognized as a robust system against parameter variations. 2) The DNNs are used to be only deployed on the high-performance computing system (e.g., CPUs, GPUs, and other accelerators [33, 1, 30]), which normally contains a variety of methods ensuring data integrity. Thus, attacking the parameters is more related to a system cyber-security topic. However, the game has been changed during the past few years. First, the robustness of the neural network to small perturbation has been put into the spotlight by adversarial examples on DNN input [11, 26]. Second, with the aid of DNN compression techniques (e.g., pruning[13] and quantization [39]) and outstanding compact neural network architectures [18, 32], deep neural networks now are friendly to the resource-limited mobile device as well. Such resource-limited platforms normally lack effective data integrity check mechanism, which makes the deployed DNN vulnerable to popular fault injection techniques, such as row hammer and laser beam [3].

Recently, there exist a cohort of works [25, 5] in an attempt to attack DNN network parameters stored in DRAM using Row Hammer Attack (RHA). However, the key limitation to these previous attack methods is that they primarily focused on extremely vulnerable full-precision DNN model (i.e., parameters in floating-point format). Our conducted simulation shows that randomly flipping the exponent part of floating-point weight could easily overwhelm the functionality of DNN. The explanation behind that is flipping the bits in exponent part of floating-point value can increase the weight to an extremely large value, thus leading to the exploded output. As a result, attacking the weight constrained DNN (i.e., weights quantized into fixed-point values) is the primary focus in this work, where the range of weight magnitude relies on the bit-width of weights.

Overview of Bit-Flip attack: In this work, we attempt to perform parameter attack on the weights of quantized DNN, 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) 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. Our proposed PBS combines gradient ranking and progressive search to locate the most vulnerable bits, while BFA performs the bit-flip operations on the located bits along their gradient ascending directions. To identify the vulnerable bits to be flipped within the identical layer and across different layers, we perform the in-layer search and cross-layer search in an iterative way. Thus, for each BFA iteration, only the most vulnerable bit elected by the PBS technique will be flip to its opposite binary value. The extensive experiments are conducted regarding various network structure, different datasets and quantization bit-width, etc. It is shocking to notice that ResNet-18 will become a random output generator (i.e., 0.1% top-1 accuracy) with only 13 bit-flips out of 93 million bits by our proposed attacking method, on ImageNet dataset.

2. Related Work

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.*, [19] 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 DRAM memory cell by just flipping one bit at a time. [31] 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) [8] and Intel SGX [12], 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 challenge becomes more severe because DNN powered applications are widely deployed in many resource-limited (e.g. smart IoT devices, mobile system, edge devices, etc.) system that lacks necessary data integrity check mechanism.

Previous Neural Network Parameter Attack. Adversarial example attack has been widely explored [38] 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 [7, 24]. 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 [25]. 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 [25]. Other works have injected faults into the activation function of the neural network to miss classify a target input [5].

Limitations of previous 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 IEEE standard for floating-point arithmetic [17]), 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 4.4. 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 ImageNet dataset.

Why we need a bit search algorithm. On the other side, most of recent deep neural network applications are performed in quantized platform such as google's Tensor Processing Unit (TPU) [37], that uses 8-bit operations for quantized network. Such fixed precision models are more robust to network parameter perturbation. Similarly, we conducted another experiment to randomly choose quantized weight for bit-flip attack using RHA. The simulation results in figure 4 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.

3. Approach

In this section, we present a novel Bit-Flip Attack (BFA) method to maliciously cause a DNN system malfunction through flipping 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 minimum number of bit-flips. It is worth to note that this work focuses on BFA on a more robust DNN with quantized weight parameters instead of floating-point number weights as discussed earlier.

3.1. Problem Definition

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[1], which is denoted as $\mathbf{B}$ in this work. More details of weights quantization are described in Section 3.2. 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 optimization problem as:

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

where $\boldsymbol{x}$ and $\boldsymbol{t}$ are the vectorized input and target output[2]. 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_b is maximum Hamming distance allowed through the entire DNN.

3.2. Quantization and Encoding

Weight quantization. In this work, we adopt a layer-wise N_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:

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

$$\mathbf{W}_l=\text{round}(\mathbf{W}_l^{\text{fp}}/\Delta w_l)\cdot\Delta w_l \tag{3}$$

where d is the dimension of weight tensor, Δw_l is the step size of weight quantizer. For training the quantized DNN with non-differential stair-case function (in Eq. (3)), we use the straight-through estimator [4] 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:

$$w/\Delta w=g(\boldsymbol{b})=-2^{N_q-1}\cdot b_{N_q-1}+\sum_{i=0}^{N_q-2}2^i\cdot b_i \tag{4}$$

With the conversion relation described by $g(\cdot)$ in Eq. (4), we can inversely obtain the binary representation of weights $\mathbf{B}$ from its fixed-point counterpart as well.

3.3. Bit-Flip Attack

In this work, we perform the BFA utilizing the similar mechanism as FGSM [11], 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. (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:

$$\nabla_{\boldsymbol{b}}\mathcal{L}=[\frac{\partial\mathcal{L}}{\partial b_{N_q-1}},...,\frac{\partial\mathcal{L}}{\partial b_0}] \tag{5}$$

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) and get perturbed bits as:

$$\hat{\boldsymbol{b}}=\boldsymbol{b}+\text{sign}(\nabla_{\boldsymbol{b}}\mathcal{L}) \tag{6}$$

where $\text{sign}(\nabla_{\boldsymbol{b}}\mathcal{L})\in\{-1,+1\}^{N_q}$. However, since the bit value is constrained between 0 and 1 ($\boldsymbol{b}\in\{0,1\}^{N_q}$), flipping the bit as Eq. (6) could lead to data overflow. Ideally, the BFA is supposed to follow the truth table in Table 1. Thus, we mathematically redefine the BFA as follows:

$$\boldsymbol{m}=\boldsymbol{b}\oplus\big(\text{sign}(\nabla_{\boldsymbol{b}}\mathcal{L})/2+0.5\big) \tag{7}$$

$$\hat{\boldsymbol{b}}=\boldsymbol{b}\oplus\boldsymbol{m} \tag{8}$$

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

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

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

Table 1. 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

3.4. 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_b most vulnerable bits per BFA iteration ($n_b = 1$ by default), thus progressively 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. 1, 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_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_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 $n_{\rm b}$ 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-$n_{\rm b}$, such process can be written as:

$$\hat{\boldsymbol{b}}_l^{k-1} = \underset{n_{\rm b}}{\rm Top} \left| \nabla_{\hat{\mathbf{B}}_l^{k-1}} \mathcal{L}\big(f(\boldsymbol{x}; \{\hat{\mathbf{B}}_l^{k-1}\}_{l=1}^L), \boldsymbol{t}\big) \right| \tag{9}$$

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

$$\hat{\boldsymbol{b}}_l^k = \hat{\boldsymbol{b}}_l^{k-1} \oplus \boldsymbol{m} \tag{10}$$

where the mask $\boldsymbol{m}$ is generated following Eq. (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. (10), which can be written as:

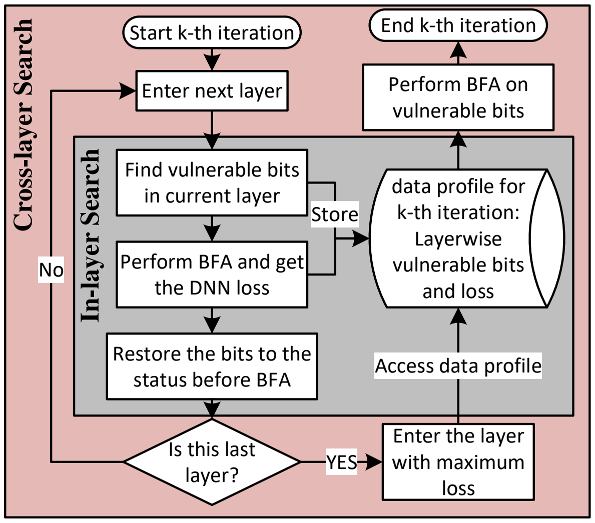
Cross-layer Search
Start k-th iteration
End k-th iteration
Enter next layer
Perform BFA on vulnerable bits
In-layer Search
Find vulnerable bits in current layer
Store
data profile for k-th iteration: Layerwise vulnerable bits and loss
Perform BFA and get the DNN loss
No
Restore the bits to the status before BFA
Access data profile
Is this last layer?
YES
Enter the layer with maximum loss

Figure 1. Flowchart to perform Progressive Bit Search (PBS) with in-layer and cross-layer search.

$$\mathcal{L}_l^k = \mathcal{L}\big(f(\boldsymbol{x}; \{\hat{\mathbf{B}}_l^k\}_{l=1}^L), \boldsymbol{t}\big) \tag{11}$$

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. (10). Note that, those bits flipped to $\hat{\boldsymbol{b}}_l^k$ in Eq. (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:

$$\begin{gathered} \hat{\boldsymbol{b}}_j^k = \hat{\boldsymbol{b}}_j^{k-1} \oplus \boldsymbol{m} \\ s.t. \;\; j = \arg\max_l \; \{\mathcal{L}_l^k\}_{l=1}^L \end{gathered} \tag{12}$$

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

4. Experiments

4.1. Experimental setup

Datasets: We take two visual datasets: CIFAR-10 [20] and ImageNet [21] for object classification task. CIFAR-10 contains 60K RGB images in size of 32×32. Following the standard practice, 50K examples are used for training and the remaining 10K for testing. The images are drawn

Table 2. BFA on CIFAR-10 with ResNet-20/32/44/56, under various quantization bit-width (N_q=4/6/8). N_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_B is the hamming distance between clean- and perturbed- binary weight ($D_\mathrm{B} = \sum_{i=1}^{L} \mathcal{D}(\hat{\mathbf{B}}_l, \mathbf{B}_l)$). The bold number with underline highlight the mismatch between two corresponding N_flip and D_B, which indicates there exist even bit-flips on the identical bit/bits.

	Baseline Acc.	$N_\mathrm{q}=8$			$N_\mathrm{q}=6$			$N_\mathrm{q}=4$		
		Acc.	N_flip	D_B	Acc.	N_flip	D_B	Acc.	N_flip	D_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]

evenly from 10 classes. ImageNet dataset contains 1.2M training images divided into 1000 distinct classes. The data augmentation used in this work is identical to methods in [15]. 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 residual network (ResNet-20/32/44/56)[15], 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_q=8), directly quantizing the pre-trained full-precision DNN without retraining (i.e., fine-tuning) only shows negligible accuracy degradation. Therefore, for fast evaluation of our proposed BFA on 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 [11] to the network. Such adversarial attack can be grouped into two major categories: 1) white-box attack [11, 26], where the adversary has full access to the network architecture and parameters, and 2) black-box attack [6, 28], 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 white box attack setup, the attacker has no access to the training dataset, training algorithm and hyper parameters used during the training of the network.

4.2. 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 2. 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 the accuracy improvement when model training is over-fitting.

Since CIFAR-10 dataset has 10 different classes of object, degrading the model's accuracy down to 10% is equivalent to make the model as random output generator. In contrast to adversarial example (e.g., PGD attack [26]), our proposed BFA is unable to degrade the network accuracy to 0%. The reason is adversarial input example attack is an input-specific attack which 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_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 2, for all the ResNet architecture with varying quantization bit-width, the required number of bit-flips N_flip to make the DNN malfunction is most likely below 20. Besides N_flip, we take the hamming distance D_B between clean- and perturbed-model as another measurable indicator. The intuition behind 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., robustness of DNN against BFA).

4.3. BFA on ImageNet

The summary of evaluation of our attack on ImageNet dataset is presented in table 3. 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 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_{flip} required to cause the accuracy to degrade to below 0.2% would be sufficient to prove the attack's effectiveness.

Table 3. BFA on ImageNet with various network architecture, under direct 8-bit weight quantization (without retraining). Accuracy (Acc.) is in top1/top5 format. N_{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_{B} is the corresponding hamming distance. Capacity is the number of bits used for weight storage (# of weights × 8).

Model (Capacity)	Baseline Acc. %	Quantized Acc. %	N_{flip}	D_{B}
AlexNet [21] (488,806,720)	56.55/79.08	56.13/78.94	17	17
ResNet-18 [15] (93,516,096)	69.76/89.08	69.50/88.98	13	13
ResNet-34 [15] (174,381,376)	73.30/91.42	73.13/91.38	11	11
ResNet-50 [15] (204,456,256)	76.15/92.87	75.84/92.82	11	11

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_{flip} decreases even more as we perform the attack on ResNet architectures. Figure 3 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 lesser number of N_{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 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.

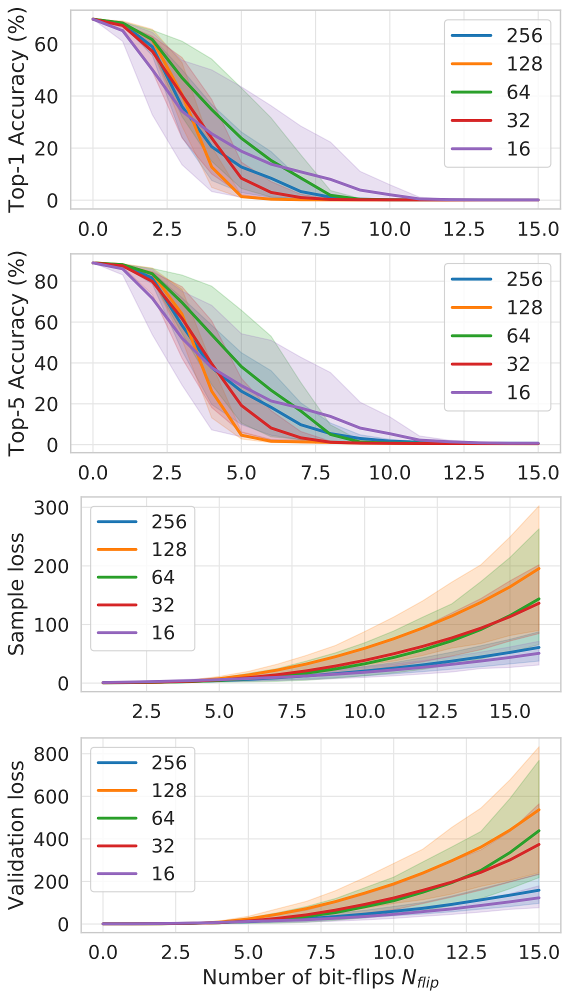

Figure 2. The BFA performance of ResNet-18 with various attack sample size (16/32/64/128/256) on ImageNet dataset. Regions in shadow indicates the error band w.r.t 5 trials.

4.4. 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 it 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. In figure 2, 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

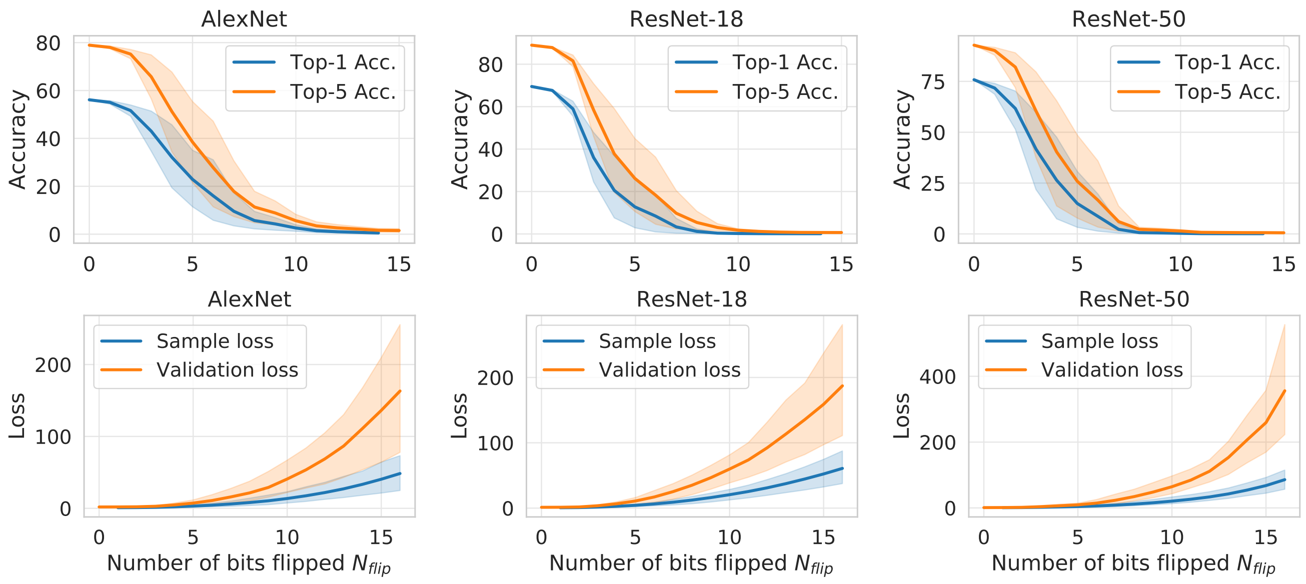

Figure 3. The accuracy (Top1/Top5) and loss evolution curve versus the number of bit-flips (N_{flip}) under BFA, for AlexNet/ResNet-18/ResNet-50 on ImageNet dataset. The sample size for performing BFA is 256. On each network architecture, we run 5 experiments and the region in shadow indicate the error-band. For all experiments in this figure, there exists no bit flipped multiple times during the attack (i.e., $N_{\text{flip}} = D_{\text{B}}$).

small input sample size, PBS still works with very small bit-flips.

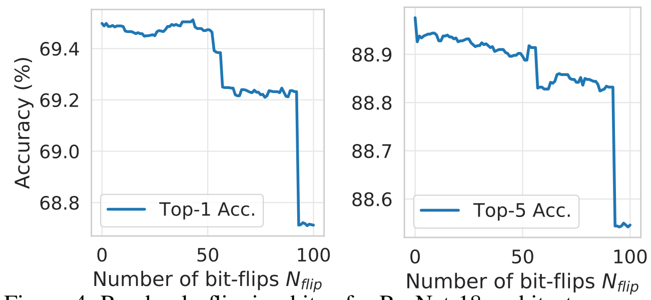

Figure 4. Randomly flipping bits of a ResNet-18 architecture on ImageNet. Even after flipping 100 random bits the network's both Top-1 and Top-5 accuracy does not degrade significantly.

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 flip on a full-precision weight(i.e, floating-point) on 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.

Then, we implement the random bit flip on 8-bit Quantized ResNet-18 architecture as shown in figure 4. It shows that by flipping even 100 random bits, the Top-1 accuracy on 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 ImageNet dataset.

4.5. 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 [25, 5] on floating-point model parameters are not efficient. They do not consider that attacking floating-point DNN model is as easy as flipping 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 neural network more resilient and robust.

5. Discussion

Why only a few bit flips can cause such destructive phenomena? In the analysis of the existence of adversary in deep neural network, Goodfellow et al. [11] concluded that deep neural networks exhibit vulnerability to adversarial examples due to their extreme linearity. The linearity of these models is the reason why they cannot resist adversary. The theory suggests that, with sufficient large input dimension, a network will always be vulnerable to noise injected at any layer. Our proposed BFA with PBS attack also introduces noise at different layers of the DNN. Any noise injected at the intermediate layer will increase as it is multiplied by the input features.

Table 4. Attacking a VGG16 [35] model's only the first and last layer separately on CIFAR-10 dataset. Attacking the first layer is much more effective. The noise injected at the early stages of the network keeps growing as it propagates through the following layers.

Layer to attack	N_{flip}	Accuracy (%)
First Conv. layer	20	10.06
Last linear layer	20	84.61

For VGG16 network we observed similar phenomena where among the 15 bit flips required to degrade the accuracy to 10 percent, 9 of them are in the first six layers. Additionally, we confirm this hypothesis of noise propagation across layers by the experiment shown in table 4. We attack the model by freezing all the layers (making them not accessible to the attacker) except the first layer, then we do the opposite by freezing all the layers except the last one. As expected, attacking the first layer achieves higher attack success. However, this linearity theory may be too simple to explain other complex phenomena inside a DNN and may not hold across different architectures. For example, ResNet architecture which has skip connections, tend to evenly distribute the bit flips across different layers.

Time Complexity of BFA with PBS. For each iteration of PBS to identify single most vulnerable bit, the time complexity is $O(L \cdot N)$, where L is the number of total convolution and linear layers, and N is the number of bits with highest gradient ranking that will be checked in PBS method. In general, the time complexity of the proposed PBS is linear for each search iteration.

BFA with PBS does not suffer from gradient obfuscation. Generation of adversarial examples in quantized network using straight-through estimator introduces gradient obfuscation [2, 23]. Attacking a quantized network becomes tricky as such network shows signs of gradient scattering [2]. In this work, we also used a quantized network which implements a uniform quantizer. However, our network directly uses quantized weights to do the inference after training. We calculate the gradient directly with respect to the quantized weights to avoid gradient obfuscation. Moreover, the performance of BFA against 4,6,8 bits quantized networks proves that the effectiveness of BFA does not degrade due to the presence of a non-differentiable function at the forward path.

Potential Defense Methods. To defend adversarial examples, the most common approach nowadays is to train the network with a mixture of clean and adversarial examples [11, 26]. One of the proposed defense methods against BFA would be to train the network to solve Madry's Min-Max optimization problem [26]. Their approach called adversarial training minimizes two losses: one from the real image and other from the adversarial image. Hence, we perform adversarial training using BFA with PBS to minimize two such losses: one computed from the original network and the other computed from the same network with one bit flip for each batch.

However, unlike adversarial training, such a training method does not help in improving the robustness of the network. Our attack can bypass adversarial training scheme primarily because of a large search space of close to 93M bits. Even if we train the network to be resilient to several bit-flips, there will always remain some bits that will be vulnerable to attack. Another potential defense against BFA can be quantized networks. Again our observation in table 2, does not show any co-relation between the number of quantization bits with the number of bit-flips required. Thus some of the popular adversarial defense methods [26, 23] fail against our BFA attack.

Data integrity check with a follow-up error correction is an ultimate solution to ensure no bits could be maliciously flipped, but it is very expensive to protect all bits (e.g., even NVIDIA 1080Ti GPU does not have ECC function). Additionally, some recent row hammer attack based methods [8, 12] can even by-pass major integrity checks, such as ECC and Intel's SGX. The above observations make our attack even more threatening for deep learning applications.

6. Conclusion

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 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.

References

[1] Shaahin Angizi, Zhezhi He, Adnan Siraj Rakin, and Deliang Fan. Cmp-pim: an energy-efficient comparator-based processing-in-memory neural network accelerator. In *Proceedings of the 55th Annual Design Automation Conference*, page 105. ACM, 2018. 1

[2] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*, 2018. 8

[3] Alessandro Barenghi, Luca Breveglieri, Israel Koren, and David Naccache. Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. *Proceedings of the IEEE*, 100(11):3056–3076, 2012. 1

[4] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. 3

[5] J Breier, X Hou, D Jap, L Ma, S Bhasin, and Y Liu. Deeplaser: Practical fault attack on deep neural networks. *ArXiv e-prints*, 2018. 1, 2, 7

[6] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui 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*, pages 15–26. ACM, 2017. 5

[7] Joseph Clements and Yingjie Lao. Hardware trojan attacks on neural networks. *arXiv preprint arXiv:1806.05768*, 2018. 2

[8] Lucian Cojocar, Kaveh Razavi, Cristiano Giuffrida, and Herbert Bos. Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks. 2, 8

[9] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015. 1

[10] Timnit Gebru, Jonathan Krause, Yilun Wang, Duyun Chen, Jia Deng, Erez Lieberman Aiden, and Li 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*, 114(50):13108–13113, 2017. 1

[11] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014. 1, 3, 5, 8

[12] Daniel Gruss, Moritz Lipp, Michael Schwarz, Daniel Genkin, Jonas Juffinger, Sioli O'Connell, Wolfgang Schoechl, and Yuval Yarom. Another flip in the wall of rowhammer defenses. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 245–261. IEEE, 2018. 2, 8

[13] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 1

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. 1

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 5, 6

[16] Zhezhi He, Adnan Siraj Rakin, and Deliang 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*, pages 588–597, 2019. 1

[17] Steve Hollasch. Ieee standard 754 floating point numbers. *Poslední změna*, 24(2), 2005. 2

[18] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016. 1

[19] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. In *ACM SIGARCH Computer Architecture News*, volume 42, pages 361–372. IEEE Press, 2014. 2

[20] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). *URL http://www. cs. toronto. edu/kriz/cifar. html*, 2010. 4

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 4, 6

[22] Fangzhou Liao, Ming Liang, Yinpeng Dong, Tianyu Pang, Jun Zhu, and Xiaolin Hu. Defense against adversarial attacks using high-level representation guided denoiser. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1778–1787, 2018. 1

[23] Ji Lin, Chuang Gan, and Song Han. Defensive quantization: When efficiency meets robustness. In *International Conference on Learning Representations*, 2019. 8

[24] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. 2017. 2

[25] Yannan Liu, Lingxiao Wei, Bo Luo, and Qiang Xu. Fault injection attack on deep neural network. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 131–138. IEEE, 2017. 1, 2, 7

[26] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018. 1, 5, 8

[27] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016. 1

[28] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practi-

cal black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 506–519. ACM, 2017. 5

[29] Aaditya Prakash, Nick Moran, Solomon Garber, Antonella DiLillo, and James Storer. Deflecting adversarial attacks with pixel deflection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8571–8580, 2018. 1

[30] Adnan Siraj Rakin, Shaahin Angizi, Zhezhi He, and Deliang Fan. Pim-tgan: A processing-in-memory accelerator for ternary generative adversarial networks. In *2018 IEEE 36th International Conference on Computer Design (ICCD)*, pages 266–273. IEEE, 2018. 1

[31] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. Flip feng shui: Hammering a needle in the software stack. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 1–18, 2016. 2

[32] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018. 1

[33] Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A Kozuch, Onur Mutlu, Phillip B Gibbons, and Todd C Mowry. Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 273–287. ACM, 2017. 1

[34] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017. 1

[35] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 8

[36] Zhun Sun, Mete Ozay, Yan Zhang, Xing Liu, and Takayuki Okatani. Feature quantization for defending against distortion of images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7957–7966, 2018. 1

[37] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016. 2

[38] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE transactions on neural networks and learning systems*, 2019. 2

[39] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016. 1, 3