On Functional Test Generation for Deep Neural Network IPs

Bo Luo, Yu Li, Lingxiao Wei and Qiang Xu
Department of Computer Science & Engineering
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
Email: {boluo,yuli,lxwei,qxu}@cse.cuhk.edu.hk

***Abstract*—Machine learning systems based on deep neural networks (DNNs) produce state-of-the-art results in many applications. Considering the large amount of training data and know-how required to generate the network, it is more practical to use third-party DNN intellectual property (IP) cores for many designs. No doubt to say, it is essential for DNN IP vendors to provide test cases for functional validation without leaking their parameters to IP users. To satisfy this requirement, we propose to effectively generate test cases that activate parameters as many as possible and propagate their perturbations to outputs. Then the functionality of DNN IPs can be validated by only checking their outputs. However, it is difficult considering large numbers of parameters and highly non-linearity of DNNs. In this paper, we tackle this problem by judiciously selecting samples from the DNN training set and applying a gradient-based method to generate new test cases. Experimental results demonstrate the efficacy of our proposed solution.**

I. Introduction

Artificial intelligence (AI) systems based on deep neural networks (DNNs) have achieved great success in many areas such as computer vision, speech recognition and natural language processing. Over the years, neural networks become increasingly larger and deeper, which requires significant amount of data and time to train. For example, it may take weeks to train a state-of-the-art model with the latest GPUs on the ImageNet dataset [1]. Consequently, it is more practical for individual users or small firms to use a trained DNN intellectual property (IP) (e.g., face recognition module) that is commercially available. In most cases, vendors would prefer a blackbox IP model to protect the architecture and the trained parameters of the DNN.

DNNs, however, are susceptible to various kinds of attacks. Adversarial example attacks [2]–[4] target to change the outputs of DNNs by slightly perturbing their inputs. Recently, there is an increasing number of attacks that target at DNNs themselves instead of their input data. Liu *et al*. [5] first proposes to attack DNN parameters for misclassifications based on two fault injection methods: single bias attack and gradient descent attack. Reverse-engineering attacks [6], [7] on hardware DNN accelerators can identify the model parameters in the off-chip memory and then attackers may stealthily substitute original parameters with malicious ones. These attacks seriously threat safety-critical applications based on DNNs. Therefore, it is essential for IP users to validate the functionality of DNNs before everyday usage.

Traditional integrity checking methods [8], [9] based on generating signatures are not applicable for DNN IPs, because IP users can not directly get the model parameters for signature generation. Hardware testing techniques for troubleshooting design defects [10], [11] are not applicable either, as IP users have no access to intermediate results of DNNs. To tackle the above problem, in this work, we propose a practical validation scheme for IP users considering their limited black-box access. The idea is for IP vendors to generate functional tests to activate parameters in the DNN whose perturbations will propagate to the outputs. Then, malicious perturbations of model parameters can be directly detected by IP users, just checking the outputs of the functional tests.

However, DNNs are highly generalized models and use non-linear activation functions, only partial parameters will be activated and take effect in the calculation for an input sample [12], thus one functional test can only validate part of parameters. Considering the large number of parameters in today's DNNs, it is challenging to generate a reasonable size of functional tests to achieve a high validation coverage. In this paper, we solve this problem with two techniques: first, we judiciously select test cases from the existing training set, and when this method becomes inefficient, a novel gradient-based technique is presented to generate new test cases. Experimental results show that the proposed functional test generation method is effective and efficient, achieving a high validation coverage with limited test cases, under both malicious and random perturbations of DNN parameters.

To the best of our knowledge, this is the first work for functional validation of DNN IPs considering end users black-box access. The main contributions of this work include:

- We formulate the functional validation of DNN IPs as an optimization problem, wherein we try to generate a small number of test cases that can activate as many parameters as possible.
- We propose to judiciously select functional tests from the training set in an iterative manner to efficiently activate DNN parameters.
- We present to generate new functional tests with a novel gradient-based method when selecting from the training set is inefficient.

The rest of the paper is organized as follows. In Section II, we give a preliminary introduction about neural networks and the related work. Then we give an overview of our functional test generation scheme in Section III. Next, the proposed efficient functional test generation method is introduced in Section IV. Finally, we present the experimental results and conclude our work in Section V and Section VI, respectively.

978-3-9819263-2-3/DATE19/©2019 EDAA

II. Preliminaries

A. Neural Networks

Neural networks are organized as successive layers of neurons which are connected by links with different parameters. Each neuron in the hidden layer applies a non-linear activation function on the weighted sum of its input. The output of layer $l+1$ is denoted as:

$$o^{(l+1)} = \sigma(W^{(l)}o^{(l)} + b^{(l)}), \quad (1)$$

where σ is the activation function. $o^{(l)}$, $W^{(l)}$ and $b^{(l)}$ are the outputs, weights and bias of the l-th layer, respectively. Weights and bias are called parameters of the network. In this way, the outputs of the current layer are computed by a non-linear function applied on the outputs of the previous layer and its parameters. Usually, there are many layers in DNNs to achieve high generality. Therefore, DNN as a whole is a complex non-linear function of parameters and the input.

Activation functions provide non-linearity so that neural networks can approximate arbitrary functions. There are several activation functions in modern neural networks, such as ReLU and Tanh, which both have some regions of saturation or inactivation [13], [14]. For example, the output of ReLU will always be zero as long as its input is negative. As neural networks are trained to fit the large training set where the training samples vary a lot to each other, an input sample can only activate partial parameters in a well trained model [12].

B. Related Work and Motivation

Past work has introduced several ways to inject faults into DNNs themselves for compromising their functionality. In [5], attackers fool DNNs to make mistakes by modifying their parameters through fault injection, in which single bias attack modifies one parameter with a large perturbation for misclassification and gradient descent attack considers stealthiness by adding small perturbations on a number of parameters. Reverse-engineering attacks [6] can identify model parameters in the off-chip memory, which may be stealthily replaced by attackers. [15] performs practical laser fault injection on activation functions of DNNs using a near-infrared diode plus laser.

To the best of our knowledge, there exists few work defending against the above functionality compromised attacks for DNN IPs. Traditional signature-based integrity checking methods [8], [9] are not applicable as IP users can not access the DNN parameters. Testing techniques [10], [11], [16], [17] generate test cases to cover all neurons so that design defects of hardware DNNs can be detected and located. However, they are not appropriate for functional validation of DNN IPs under attacks for two reasons: first, IP users have no access to the intermediate model results as system testers do. Second, testing only considers the neuron coverage, which is not enough for covering model parameters under malicious attacks. For example, there are two neurons in adjacent layers that are covered by two separate test cases and no other tests cover them during the testing process. Even though the two neurons can be tested, the attacks targeting to perturb the weight between them cannot be detected. As the two neurons are never activated at the same time with test cases, the malicious perturbations on the weight will never be revealed, but it may cause misclassifications for other inputs.

Motivated by the above, in this paper, we propose to validate the functionality of DNN IPs by effectively generating a small number of test cases that can activate model parameters whenever possible and propagate their perturbations to the outputs. IP users only have to run these test cases and check the final outputs of DNNs to validate their functionality without knowing model details. To the best of our knowledge, this is the first work of functional validation for DNN IPs under malicious attacks targeting at model parameters, as detailed in the following sections.

III. DNN IPs Validation Methodology

As discussed in previous sections, IP users can just use the DNN IP as a black box: feed the IP with an input and get the corresponding output. Based on this, we propose a practical functional validation scheme for IP users, in which IP vendors will first generate a small number of functional tests and share them with IP users, then users validate the functionality of the IP by checking whether it functions correctly with the shared tests.

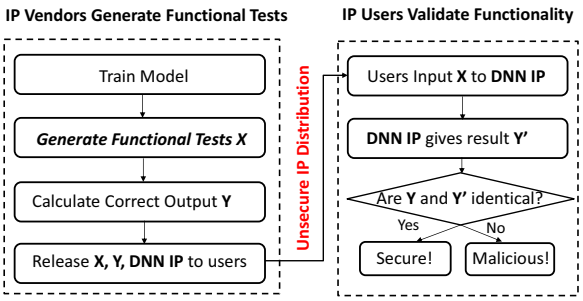

Fig. 1: The overview of functional validation for DNN IPs.

The working flow of the proposed functional validation scheme is shown in Fig. 1, which consists of two phases. Firstly, for the IP vendors, they generate a small set of functional tests X, then release these test cases, the corresponding outputs Y together with the IP to users. After through an unsecure distribution process, IP users receive the IP and run it as a black box with these functional tests X. Then they compare the current outputs Y' with the provided ones Y. If they are not identical, the DNN IP has been perturbed. Otherwise, it is secure. The shared functional tests X and the corresponding outputs Y are encrypted, thus their integrity can be ensured.

As DNNs are extremely complex and non-linear, only partial parameters take effect for one test case. The perturbations of other parameters will not be detected and thus not be validated with this test. Therefore, the key challenge in our validation scheme is to effectively generate a reasonable number of functional tests that can validate DNN parameters

as many as possible under malicious perturbations, which is demonstrated in the next section.

IV. Efficient Functional Test Generation

In this section, we first define our validation objective. Then we propose to judiciously select tests from the existing training set and when this method becomes inefficient, a new gradient-based test generation technique is presented. Finally, these two approaches are combined in a unified way to efficiently generate functional tests for DNN IPs.

A. Validation Objective

In our validation scheme, we call a parameter is activated when perturbations of it will propagate to the DNN output and be detected. Otherwise, it is un-activated. A parameter can be validated when at least one test case activates it. As the gradient of a function measures the sensitivity of its output with respect to the change of its argument, we use the gradient of the DNN output with respect to the parameter to determine whether the parameter is activated or not. Assume ReLU is the activation function, given an input $\boldsymbol{x}$, we define the parameter θ_i is activated if it satisfies:

$$\nabla_{\theta_i} F(\boldsymbol{x}) \neq 0, \tag{2}$$

where F is the function of the DNN. $\nabla_{\theta_i} F(\boldsymbol{x})$ calculates the gradient of $F(\boldsymbol{x})$ with respect to θ_i. Unlike ReLU, the gradients of other activations (e.g., Sigmoid and Tanh) in the saturation regions are quite small and approximate to zero. In this case, we define θ_i is activated when $\nabla_{\theta_i} F(\boldsymbol{x})$ is greater than a small value ϵ. For the easy explanation of our method, we assume ReLU is the default activation function.

Therefore, the validation coverage of a functional test $\boldsymbol{x}$ can be formulated as follows:

$$VC(\boldsymbol{x}) = \frac{\#\{\theta \mid \nabla_\theta F(\boldsymbol{x}) \neq 0\}}{\#(\theta)}, \tag{3}$$

where the numerator is the number of activated parameters, and the denominator is the number of total parameters in the DNN. The validation coverage of a functional test equals to the percentage of parameters it activates.

As one functional test can only activate partial parameters, it is necessary to use a set of functional tests to achieve a high validation coverage. Given a test set X with n samples, its validation coverage is as follows:

$$VC(X) = \frac{\#(P_1 \cup P_2 \cup ...P_n)}{\#(\theta)}, \tag{4}$$

where

$$P_i = \{\theta \mid \nabla_\theta F(\boldsymbol{x}_i) \neq 0\}. \tag{5}$$

P_i denotes the parameter set activated by the test case $\boldsymbol{x}_i$, and the validation coverage of X is the percentage of unique parameters activated by all tests in X.

Generally speaking, more test cases can activate more parameters and thus obtain a higher validation coverage, but will incur a larger validation cost. Therefore, it is essential to achieve a good tradeoff between the validation coverage and cost. We formulate this problem as follows:

$$\begin{aligned} &\arg\max_X \ VC(X) \\ &s.t. \quad \#(X) \leq N_t, \end{aligned} \tag{6}$$

where N_t is the maximum number of test cases allowed for functional validation. Our objective is to maximize the validation coverage with a limited number of test cases. Next, we introduce techniques to solve this problem in detail.

B. Selecting from Training Set

The first solution we propose is to select functional tests from the existing training set based on the following heuristic: as the DNN is trained to successfully perform some tasks (e.g., regression and classification) on the training set, most parameters will participate in processing these tasks. In other words, if many parameters are not activated in the training set, the network is not trained well, as many resources are wasted.

Based on the above analysis, we judiciously select test cases from the training samples in an iterative manner. In each iteration, we choose the sample that can activate the maximal number of un-activated parameters. At the beginning, the chosen validation set is empty, and the sample with the highest validation coverage is firstly selected. Then in the following iterations, we choose the next sample $\boldsymbol{s}_i$ from the training set S according to the following equation:

$$\arg\max_{\boldsymbol{s}_i \in S} \ VC(X + \boldsymbol{s}_i) - VC(X), \tag{7}$$

where X is the current validation set that includes the chosen samples in previous iterations. This equation selects the input that can activate the most un-activated parameters or lead to the largest validation coverage increase.

Algorithm 1: Selecting from training set.

Input: DNN function F, training set S, maximum functional tests N_t.
Output: Validation set X.

```
1 Initialize validation set: X = ∅;
2 while #(X) < N_t do
3     for s_i ∈ S do
4         Δ_{s_i} VC = VC(X + s_i) − VC(X);
5     end
6     Select s_i with the largest Δ_{s_i} VC;
7     Add s_i to the validation set X;
8     Update VC(X).
9 end
```

The whole process of selecting functional tests from the training set is shown in Algorithm 1, in which we first initialize the validation set as empty. During each iteration, we calculate the benefit or the increased validation coverage $\Delta_{\boldsymbol{s}_i} VC$ achieved for each training sample in line 3-5. Then we select the best one which brings the largest validation coverage increase, and add it to the validation set X in line 7. The iteration is continued until the number of functional tests exceeds the limit N_t.

The experimental results in Section V show that this method is effective at early iterations, achieving a high

validation coverage with a very small number of functional tests. However, in late iterations, the validation coverage will increase extremely slow with new functional tests added. That is to say, the method will saturate quickly. To solve this problem, next we propose to generate new samples to activate the remaining parameters as many as possible when training samples are no longer efficient.

C. Gradient-based Test Generation

Considering there are some parameters difficult to activate by the training samples, we propose to generate new samples to activate these bottleneck parameters. The key idea is to generate *synthetic training samples* which can be classified correctly by the network consists of the un-activated parameters. The intuition is that samples correctly classified by a DNN will have similar features with its training samples, thus can efficiently activate the network parameters. Based on this, we propose to efficiently activate the bottleneck parameters by generating synthetic training samples based on the gradient descent technique widely used for training DNNs.

Unlike training DNNs, wherein parameters are updated to minimize the loss, we update the input to reduce the loss according to the gradients of it. This can be formulated as follows:

$$\boldsymbol{x}_i^* = \boldsymbol{x}_i - \eta \nabla_{\boldsymbol{x}_i} J(\boldsymbol{x}_i, y_i, \theta), \tag{8}$$

where $J(\boldsymbol{x}_i, y_i, \theta)$ is the loss function that measures the gap between the model output for an input $\boldsymbol{x}_i$ and the corresponding ground truth y_i. In each update, we change the input with the step size η at the directions based on the gradients of $J(\boldsymbol{x}_i, y_i, \theta)$ with respect to $\boldsymbol{x}_i$, in which the loss can decrease most quickly. After several iterations, we can get the synthetic training samples that can be classified correctly by the network with un-activated parameters.

In each iteration, we generate a batch of k synthetic training samples where k is the number of the neurons in the output layer. We do this because for classification, the number of neurons in the output layer corresponds to the number of classification categories. Each category has their own unique features and a batch of input containing all these categories will have a higher probability to activate more parameters.

The overall process of gradient-based test generation is shown in Algorithm 2, where in each iteration, we generate k input patterns, classified as k different categories, respectively. First, in line 3, the inputs are initialized with all zeros. Then, we update these inputs using gradient descent method to iteratively decrease the loss function J in line 5-11. After T iterations, the generated k tests can be classified by the model correctly and we add them to the validation set X in line 12. The process is continued until the number of generated functional tests reaches to the limit.

D. Combined Functional Test Generation

As Algorithm 1 is effective at early iterations but quickly becomes inefficient, Algorithm 2 can continually increase the validation coverage, but is not as efficient as Algorithm 1 in the early stage (the true training samples are more effective than the synthetic ones). Therefore, we propose to combine these two functional test generation techniques in a unified way, where we generate tests with Algorithm 1 first, and then switch to Algorithm 2 when Algorithm 1 is inefficient. However, the remained problem is to identify the switch point. We propose to compare the benefit achieved by each algorithm. When the increased validation coverage per test case generated by Algorithm 2 is greater than the one generated by Algorithm 1, we will transform to gradient-based test generation method.

Algorithm 2: Gradient-based test generation.

Input: Loss J, category number k, maximum functional tests N_t, maximum gradient descent updates T.
Output: Validation set X.

```
1  Initialize validation set: X = ∅.
2  while #(X) < N_t do
3      Initialize x_1^*, x_2^*, ... , x_k^* with all zeros;
4      t = 0;
5      while t < T do
6          for i ← 1 to k do
7              δ = η∇_{x_i^*} J(x_i^*, y_i, θ);
8              x_i^* = x_i^* − δ;
9          end
10         t = t + 1;
11     end
12     Add x_1^*, x_2^*, ... , x_k^* to X.
13 end
```

V. Experimental Results

A. Experimental Setup

The experiments are performed with MNIST [18] and CIFAR-10 [19] datasets. The MNIST includes 70000 handwritten digit images, and the CIFAR-10 contains 60000 color images of natural objects. To verify that our validation scheme can apply to varying DNN architectures and activation functions, we train the MNIST model with Tanh activation function, and the CIFAR-10 model with ReLU.

For each dataset, we implement one DNN model, detailed in Table I. The MNIST and CIFAR-10 models achieve 98.9% and 84.26% classification accuracy respectively, which are comparable to the state-of-the-art results.

Layer	MNIST	CIFAR
1	28*28 Image	32*32 RGB Image
2	Conv(3,3,32). Tanh	Conv(3,3,64). ReLU
3	Conv(3,3,32). Tanh Max pooling(2,2)	Conv(3,3,64). ReLU Max pooling(2,2)
4	Conv(3,3,64). Tanh	Conv(3,3,128). ReLU
5	Conv(3,3,64). Tanh Max pooling(2,2)	Conv(3,3,128). ReLU Max pooling(2,2)
6	Fully connect 128. Tanh	Fully connect 512. ReLU
7	Fully connect 10.	Fully connect 10.
Softmax		

TABLE I: The architectures of the two models.

B. Validation Coverage

In this section, we evaluate the validation coverage of the proposed functional test generation method.

1) Validation Coverage of Different Image Sets: Fig. 2 shows the validation coverage of three different image sets: the first one is the noisy images of Gaussian distribution; the second is the ImageNet that is the largest data set in the image recognition area [1]; the third is the training set of the corresponding model. For each image set, we randomly select 1000 images and calculate their average validation coverage.

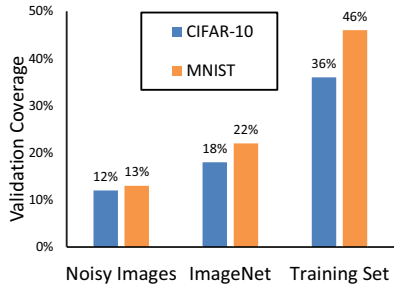

Fig. 2: Validation coverage of different image sets.

We can see that the training samples achieve the highest validation coverage for both the MNIST and CIFAR-10 model with 46% and 36%, respectively. And the ImageNet achieves the second best performance, while random images achieve the worst, where the validation coverage is only 13% for the MNIST and 12% for the CIFAR-10. The results correspond to our analysis that DNNs will take full advantage of their resources (e.g., parameters) to finish the classification task on training samples. As a result, images from the training set will have a higher probability to activate more parameters than others. Noisy images have little features similar to the training samples and thus activate the least number of parameters.

2) Validation Coverage of Different Methods: Fig. 3 shows the validation coverage of the proposed three functional test generation methods for the CIFAR-10 model, in which we can see that a small number of selected training samples can achieve a high validation coverage, for example, only 20 functional tests can obtain up to 82% validation coverage. However, selecting from training samples will become inefficient quickly. The validation coverage only increases 4% when the number of functional tests increases from 20 to 10000. Moreover, we find that there are about 8% of parameters always un-activated when using the whole training set. We analyze this phenomenon that DNNs are highly generalized models and some parameters are reserved for samples unseen in the training set.

For gradient based functional test generation, the validation coverage keeps increasing until it achieves almost 100%. This is because it can iteratively activate the un-activated parameters of DNNs by generating synthetic training samples for the remaining networks. However, it is not as efficient as selecting from training samples in the early stage, as training samples can activate more parameters than the synthetic ones. According to Fig. 3, 10 functional tests from the training set can activate about 78% parameters,

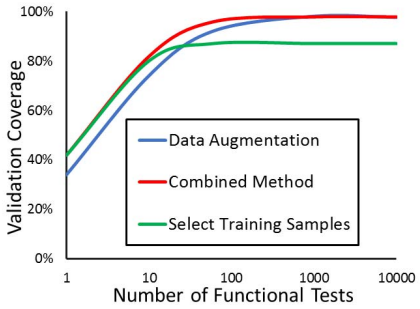

Fig. 3: Validation coverage of different methods on CIFAR.

while 10 tests generated based on gradient descent method can only activate about 66%.

Therefore, selecting tests from the training set is efficient at the early iterations, while gradient based method is efficient in the late stages. This justifies the necessity of our combined method which takes the advantages of both methods. From the red line in Fig. 3, we can see that our combined method achieves the best validation coverage and cost tradeoff, where 30 tests can activate 92% parameters, while 30 training samples or synthetic samples can just activate 84% or 76%, respectively.

Moreover, to analyze the effectiveness of synthetic training samples for activating parameters, we show the real and synthetic training samples in Fig. 4. We can see that the generated samples do share some common features with the training samples of the same category. For example, the generated digit 0 in the second row has a circle in the image which is an important feature for recognizing 0. Thus, we can conclude that our gradient-based functional test generation method can efficiently generate samples containing important features for recognition and activate parameters effectively as training samples do.

Fig. 4: training samples vs. synthetic samples of MNIST.

C. Perturbation Detection Rate

In this section, we evaluate the performance of the proposed validation scheme considering its detection rate under malicious and random parameter perturbations. The malicious perturbations are generated according to the attacks proposed in [5] and the random perturbations are to add gaussian noises. We implement each kind of parameter perturbation 10000 times against the MNIST and CIFAR-10 models, and then calculate the detection rate by observing whether the perturbations will change the DNN outputs of the generated functional tests. In order to justify the necessity of considering parameter coverage instead of neuron coverage, we compare our combined functional test

	Tests with neuron coverage			Proposed with parameter coverage		
Number of Tests	SBA	GDA	Random	SBA	GDA	Random
N=10	59.0%	67.2%	58.7%	87.2%	89.4%	86.3%
N=20	67.4%	76.5%	65.9%	91.1%	92.5%	90.4%
N=30	76.3%	84.1%	74.8%	93.5%	94.7%	92.2%
N=40	82.5%	90.2%	80.2%	95.2%	96.3%	93.6%
N=50	89.1%	92.6%	84.3%	97.3%	98.1%	96.1%

TABLE II: Detection rate under different perturbations on MNIST.

	Tests with neuron coverage			Proposed with parameter coverage		
Number of Tests	SBA	GDA	Random	SBA	GDA	Random
N=10	42.2%	53.1%	40.3%	81.0%	82.1%	79.6%
N=20	58.3%	67.2%	57.6%	87.2%	89.0%	86.2%
N=30	69.2%	76.5%	68.8%	92.2%	93.9%	90.8%
N=40	76.7%	84.8%	76.0%	94.5%	96.2%	93.2%
N=50	82.8%	90.7%	82.6%	95.7%	97.3%	95.2%

TABLE III: Detection rate under different perturbations on CIFAR.

generation method with the hardware testing technique that only considers neuron coverage [11]. It should be noted that hardware testing cannot be used in this case as users have no access to intermediate DNN results.

Table II and III show the detection rates for MNIST and CIFAR-10 under single bias attack (SBA), gradient descent attack (GDA) [5] and random perturbations, respectively. We can see that our combined test generation method achieves 87.2% and 89.0% detection rates under SBA and GDA respectively with only 20 functional tests for the CIFAR-10 model. Comparing with the test generation method considering neuron coverage, it performs worse than our combined method, achieving much lower detection rate with the same number of functional tests. Even though all neurons are covered by test cases, it is not necessarily to cover all parameters. This justifies the necessity of considering parameter coverage in our proposed solution.

VI. Conclusions

In this paper, we propose a practical validation scheme for DNN IPs without showing users model parameters. The idea is to generate a small number of functional tests to largely activate model parameters. Then the perturbations on them will propagate to the outputs and be detected. Considering the large amounts of parameters and highly non-linearity of DNNs, it is very challenging to solve this problem. In this work, we first propose to judiciously select test cases from the training set and when this method becomes inefficient, we present a gradient-based new test generation techniques. Finally, these two methods are combined in a unified way to achieve both advantages. Experimental results show that our solution achieves a good trade off between validation coverage and cost, and can effectively detect malicious and random perturbations with a reasonable number of tests.

Acknowledgement

This work was supported in part by the General Research Fund (GRF) of Hong Kong Research Grants Council (RGC) under Grant No. 14205018 and in part by National Natural Science Foundation of China under Grant No. 61432017 and No. 61532017.

References

[1] J. Deng, W. Dong, R. Socher, *et al.*, "Imagenet: A large-scale hierarchical image database," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[2] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *International Conference on Learning Representations (ICLR)*, 2015.

[3] N. Papernot, P. McDaniel, S. Jha, *et al.*, "The limitations of deep learning in adversarial settings," *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2016.

[4] B. Luo, Y. Liu, L. Wei, *et al.*, "Towards imperceptible and robust adversarial example attacks against neural networks," *AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[5] Y. Liu, L. Wei, B. Luo, *et al.*, "Fault injection attack on deep neural network," *International Conference on Computer-Aided Design (ICCAD)*, 2017.

[6] W. Hua, Z. Zhang, and G. E. Suh, "Reverse engineering convolutional neural networks through side-channel information leaks," *Design Automation Conference (DAC)*, 2018.

[7] L. Wei, B. Luo, Y. Li, *et al.*, "I know what you see: Power side-channel attack on convolutional neural network accelerators," *Annual Computer Security Applications Conference (ACSAC)*, 2018.

[8] M. Ohkubo, K. Suzuki, S. Kinoshita, *et al.*, "Cryptographic approach to privacy-friendly tags," in *RFID privacy workshop*, vol. 82, Cambridge, USA, 2003.

[9] R. Venkatesan, S.-M. Koon, M. H. Jakubowski, *et al.*, "Robust image hashing," *International Conference on Image Processing (ICIP)*, 2000.

[10] K. Pei, Y. Cao, J. Yang, *et al.*, "Deepxplore: Automated whitebox testing of deep learning systems," *ACM Symposium on Operating Systems Principles (SOSP)*, 2017.

[11] L. Ma, F. Zhang, M. Xue, *et al.*, "Combinatorial testing for deep learning systems," *arXiv preprint arXiv:1806.07723*, 2018.

[12] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

[13] J. Han and C. Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning," *International Conference on Artificial Neural Networks (ICANN)*, 1995.

[14] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," *International Conference on Machine Learning (ICML)*, 2010.

[15] J. Breier, X. Hou, D. Jap, *et al.*, "Practical fault attack on deep neural networks," *arXiv preprint arXiv:1806.05859*, 2018.

[16] Y. Sun, X. Huang, and D. Kroening, "Testing deep neural networks," *arXiv preprint arXiv:1803.04792*, 2018.

[17] T. Gehr, M. Mirman, D. Drachsler-Cohen, *et al.*, "Ai 2: Safety and robustness certification of neural networks with abstract interpretation," *IEEE Symposium on Security and Privacy (S&P)*, 2018.

[18] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *http://yann. lecun. com/exdb/mnist*, 2010.

[19] A. Krizhevsky, V. Nair, and G. Hinton, "The cifar-10 dataset," *http://www. cs. toronto. edu/kriz/cifar*, 2014.