This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2020.2997359, IEEE Transactions on Network Science and Engineering

Coverage Guided Differential Adversarial Testing of Deep Learning Systems

Jianmin Guo, Yue Zhao, Houbing Song, and Yu Jiang

***Abstract*—Deep learning is increasingly applied to safety-critical application domains such as autonomous cars and medical devices. It is of significant importance to ensure their reliability and robustness. In this paper, we propose DLFuzz, the coverage guided differential adversarial testing framework to guide deep learing systems exposing incorrect behaviors. DLFuzz keeps minutely mutating the input to maximize the neuron coverage and the prediction difference between the original input and the mutated input, without manual labeling effort or cross-referencing oracles from other systems with the same functionality. We also design multiple novel strategies for neuron selection to improve the neuron coverage. The incorrect behaviors obtained by DLFuzz are then exploited for retraining and improving the dependability of the models.**

We present empirical evaluations on two well-known datasets to demonstrate its effectiveness. Compared with DeepXplore, the state-of-the-art deep learning white-box testing framework, DLFuzz does not require extra efforts to find similar functional deep learning systems for cross-referencing check. But DLFuzz could generate 338.59% more adversarial inputs with 89.82% smaller perturbations, while maintaining the identities of the original inputs. DLFuzz also managed to averagely obtain 2.86% higher neuron coverage, and save 20.11% time consumption with respect to DeepXplore. We then evaluate the effectiveness of strategies for neuron selection, and demonstrated that all these strategies perform better than DeepXplore. Finally, DLFuzz proved to be able to improve the accuracy of deep learning systems by incorporating these adversarial inputs to retrain.

***Index Terms*—deep learning, dependability, adversarial testing, neuron coverage.**

I. Introduction

DEEP learning plays a key role in the development of artificial intelligence. In the past few years, deep learning has demonstrated its competitiveness on a wide range of applications, such as image classification [1], [2] in autonomous cars, natural language processing [3] in robotics and even reconstruction of brain circuits [4] in medical devices. These encouraging accomplishments inspired wide deployments of deep learning techniques in more security-critical domains, and it is in great demand to test their dependability.

For the testing of deep learning systems, the classical approach is to gather sufficient manually labeled input data to assess the accuracy. However, the input space of testing is so huge that it is extremely hard to collect all the possible inputs to trigger every feasible logic of a deep learning system. It is demonstrated that state-of-the-art deep learning systems can be fooled by adding small perturbations to the test inputs [5]. Although deep learning exhibits impressive performance on image classification or recognition tasks, the classifiers can also be easily led to incorrect classifications by applying imperceptible perturbations [6], as shown in Fig. 3. Therefore, deep learning systems testing is quite challenging but essential to ensure the correctness of those safety-critical practices.

Several approaches have been proposed to improve the testing efficiency of deep learning systems with software testing techniques. Some of them leverage solvers like Z3 to generate adversarial inputs under the formalized constraints of the deep learning models [7], [8]. These techniques are accurate, but work in a heavy white-box manner and are resource-consuming for constraint solving. Some black-box methods exploit heuristic algorithms to mutate the inputs until the adversarial inputs acquired [9]. These methods are time-consuming and rely heavily on the manually supplied ground truth labels. Other approaches of adversarial deep learning focus on fooling the deep learning systems by applying imperceptible perturbations to the inputs mostly in a gradient-based manner [5], [6]. They work efficiently but are shown to have low neuron coverage [10].

Recently, DeepXplore [10] was presented as the state-of-the-art white-box testing framework for deep learning systems and first introduced the concept of neuron coverage as a testing metric. It relies on three deep learning systems with similar functionality to cross-check and finds adversarial inputs which produce different labels among these systems. Given one test input, DeepXplore maximizes the difference of labels of tested three systems and value of a neuron selected to cover jointly. It could generate adversarial inputs efficiently and improve neuron coverage substantially with respect to random testing methods. Nevertheless, cross-referencing suffers from the scalability and difficulty of finding similar deep learning systems.

Owing to the totally distinct internal structures of deep neural networks (DNN) and software programs, there exists a large gap between the testing of deep learning systems and the testing of traditional software systems. More efforts are needed for approaches which could better combine software testing into deep learning testing. Fuzz testing [11]–[13] has been recognized as one of the most effective methodologies for vulnerability detection in software testing, demonstrated by the huge amount of vulnerabilities caught. The core idea is to generate random inputs to execute as many program paths as possible so as to lead the program to expose violations and crashes. It can be seen that fuzz testing and deep learning systems testing share similar goals of achieving higher coverage as well as getting more exceptional behaviors. Furthermore, multiple strategies for input mutation play a key

J. Guo and Y. Zhao and Y. Jiang (corresponding author, email: jiangyu198964@126.com) are with the KLISS, BNRist, School of Software Engineering, Tsinghua University, Beijing, 100084 China. H. Song is with the Department of Electrical Engineering and Computer Science, Embry-Riddle Aeronautical University, Daytona Beach, FL 32114 USA (h.song@ieee.org).

part in generating high-quality inputs randomly. To achieve higher coverage, it keeps the mutated inputs contributing to the increase of coverage in a seed list. Inspired by the similar goal and useful experience of fuzz testing for later mutation, the way of combining its knowledge into the adversarial testing of deep learning systems is worth practicing.

In this paper, we propose DLFuzz, a coverage guided differential adversarial testing framework. It aims to maximize the neuron coverage and to generate more adversarial inputs for a given deep learning system, without cross-referencing other similar systems or manual labeling. First, DLFuzz iteratively selects neurons worthy to activate for covering more logics, where multiple strategies are designed based on our premier work [14]. Next, DLFuzz mutates the test inputs by applying minute perturbations to guide the deep learning system exposing incorrect behaviors. During the mutation process, DLFuzz keeps the mutated inputs which contribute to a certain increase of the neuron coverage for the subsequent fuzzing. Besides, the minute perturbation is restricted to be invisible so that the prediction results of the original input and the mutated inputs should be the same. In this way, DLFuzz is able to automatically identify the erroneous behaviors with differential testing that an error is triggered when the prediction result of the mutated input is not the same as the original input.

To evaluate the efficiency of DLFuzz, we conducted empirical studies on six deep learning systems trained on two popular datasets, MNIST [15] and ImageNet [16]. The deep learning model and the datasets are exactly the same as those used by DeepXplore. Compared with DeepXplore, DLFuzz does not need extra efforts to collect similar functional deep learning systems for cross-referencing label check, but could generate $135\% - 584.62\%$ more adversarial inputs with $79.56\% - 96.77\%$ smaller perturbation. In the different settings, DLFuzz obtained $1.10\% - 5.59\%$ higher neuron coverage. Besides, DLFuzz achieved higher neuron coverage even in all the configurations of activation threshold of DeepXplore. For the time efficiency, it saves 20.11% time consumption on average with one exceptional case on ImageNet costing 59.42% more than DeepXplore. For the practical concerns, we carried out thorough evaluations of the configurations of the parameters and recommended better choices of them. Next, we evaluate the effectiveness of strategies for neuron selection, and demonstrated that all these strategies perform better than DeepXplore while the better strategies for each deep learning system vary. Finally, DLFuzz exhibits its practical use for steadily improving the deep learning systems by augmenting the training dataset and retraining the model.

In summary, our work has the following main contributions:

1) We combine fuzz testing into the adversarial testing of deep learning systems and propose the first coverage guided differential fuzz testing framework, which provides a novel direction for the testing of deep learning based systems.
2) We leverage differential testing to avoid manually checking effort and overcome the trouble of collecting similar functional deep learning systems for cross-referencing.
3) We implement the proposed framework and conduct comprehensive experiments to demonstrate its highly effectiveness in improving the dependability of deep learning systems, compared with DeepXplore. Moreover, the code, testing dataset and experimental results are all available at https://github.com/turned2670/DLFuzz.

II. Related Work

We present the most closely related work of deep learning testing and fuzzing testing. The former two parts provide two different research directions for deep learning testing.

Adversarial deep learning. The robustness and reliability of deep learning systems are the crucial factors for their large-scale applications in the real world. However, an intriguing property of deep learning systems was found out that a state-of-art DNN can be easily fooled by applying imperceptible perturbations to the input image [5]. Since then, a large number of methodologies have emerged in this field. The classical approach FGSM [17] computes the adversarial perturbation efficiently by one-step gradient ascent. Another classical approach JSMA [18] fools the deep learning system by modifying several pixels according to a saliency map recording the input features. Many approaches extended these work and developed multiple trends including iterative methods [19], [20], nontargeted attacks and targeted attacks [6], [20], black-box attacks [21], defenses against attacks [22], attacks beyond the context of image classification [23], [24], etc. For detailed information on more related work, we recommend a comprehensive survey about adversarial attacks on deep learning [25].

As discussed earlier, these approaches perform efficiently in generating adversarial inputs for the deep learning systems as well as obtaining minute perturbations [6]. However, they expose a major weakness that they achieve low neuron coverage, similar to the coverage obtained using randomly selected test inputs [10].

Testing and verification of deep learning systems. Inspired by the achievements of software testing for ensuring the robustness of traditional programs, researchers also tried to apply these techniques to deep learning testing for ensuring the robustness of deep learning systems. White-box verification approaches aim to guarantee the safety of deep learning systems by formally checking the violations of safety properties. The early approach [26] defines safety constraints for the whole deep learning system and solves them by an SAT solver. Later works [7], [8] attempt to improve the scalability of these approaches within a finite scope. However, they still have a long way to serving the real-world deep learning systems. Black-box testing approaches [9], [27] search the input vector space for modifications which could cause errors. Even with salient regions limiting the search space relatively, these methods are still too time-consuming.

Unlike the above methods, DeepXplore [10] was proposed as the first white-box testing framework which could scale well to large-scale deep learning systems. It introduced neuron coverage as a metric similar to the code coverage in traditional testing. Several other approaches [8], [28] have also proposed multiple metrics of different granularity for deep learning testing.

Fuzz testing of software. Fuzz testing is an advanced methodology of vulnerability detection in software testing.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2020.2997359, IEEE Transactions on Network Science and Engineering

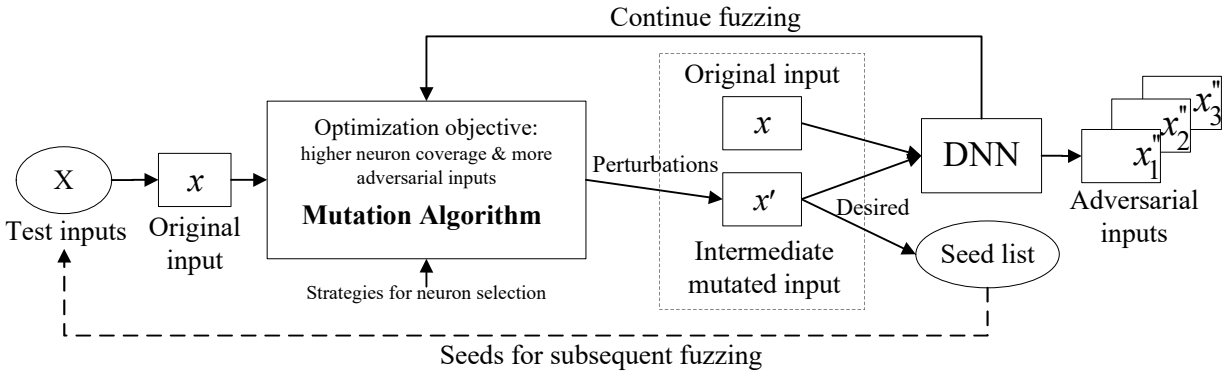


Fig. 1: Architecture of DLFuzz for the differential adversarial testing of deep learning systems.

It is well-known for its highly effectiveness in discovering vulnerabilities in real-world applications. Most of the public vulnerabilities are reported by fuzz testing tools. AFL [11] and Libfuzzer [29] are two fundamental fuzzing tools and are widely deployed in security communities and academic research. AFL tends to test the whole program while Libfuzzer focuses on fuzzing specific functions. Their key features for producing random inputs and choosing better seeds to increase the coverage have been combined into deep learning testing in this paper.

Other approaches have made attempts to improve fuzzing. Coverage-based fuzzers [12], [13], [30] leverage the coverage information to guide the mutating process for valuable seeds. Mutation-based fuzzers [31], [32] concentrate on fuzzing the complex formats using the input format model. Some tools [33], [34] have exploited symbolic execution to generate the real inputs for feasible paths satisfying the constraints. The means of utilizing these successful experience of fuzzing in deep learning testing may be worthy of exploration further.

III. DLFuzz System Design

A. DLFuzz Overview

The overall architecture of DLFuzz is depicted in Fig. 1. In this paper, we implement DLFuzz to work on image classification based deep learning systems, a popular task in deep artificial intelligence domains to demonstrate its feasibility and effectiveness. The adaptions in other tasks such as speech recognition are straightforward and also follow the same workflow in Fig. 1.

To specify, the whole test input set X is composed of images to be classified and each input x is the one during testing. The DNN is a particular convolutional neural network (CNN) under test, such as VGG-16 [35]. The mutation algorithm applies tiny perturbation to x and gets x', which is visibly indistinguishable from x. If the mutated input x' and the original input x are both fed to the CNN but classified to be of different class labels, we treat this as an incorrect behavior and x' to be one of the adversarial inputs. The inconsistent classification results before and after mutation indicate that at least one of them is wrong so that manually labeling effort is not required here. In contrast, if the two are predicted of the same class label by the CNN, x' will continue to be mutated by the mutation algorithm to test the CNN's robustness. In addition, a seed list is well maintained for each given input x, keeping those intermediate mutated inputs which could increase the neuron coverage and satisfy the limit for the perturbation. These seeds are then added to the test input set X, used for subsequent fuzzing and producing incremental adversarial inputs.

B. Coverage Definitions

Neuron coverage. Based on the demonstration that covering more neurons could potentially trigger more logics and more erroneous behaviors [10], DLFuzz also leverages the concept of neuron coverage. It refers to the percent of neurons of the DNN which have been activated at least once during testing.

The formal definition is represented in equation (1). Given a set of test inputs $T = \{x_1, x_2, ...\}$ and all neurons of the testing DNN $N = \{n_1, n_2, ...\}$, for a specific neuron $n \in N$, its output value under a specific input x fed to the DNN is denoted by $out(n, x)$. Neuron n is regarded as activated (covered) if there exists one such input x that $out(n, x)$ is larger than the set threshold t. Here the threshold t should be customized when testing.

$$NC(T) = \frac{|\{n|\exists x \in T, out(n, x) > t\}|}{|N|} \tag{1}$$

l_2 distance. DLFuzz adopts l_2 norm, one of the most frequently used norms, to measure the perturbations and restrict the perturbations to be nearly imperceptible. Considering the observation that the same minute perturbation may produce varying degrees of impact when applied to different inputs, we introduce a relative measurement denoted by l_2 distance, which is defined to be the ratio of the l_2 norm of the perturbation to the l_2 norm of the corresponding test input. Similar to [6], we also adopt the average l_2 distance of the whole test inputs computed as follows as an important metric.

$$\overline{l_2\ dist.} = \frac{1}{|T|} \sum_{x \in T} \frac{\|r\|_2}{\|x\|_2} \tag{2}$$

where T and x are the same as the definitions in equation (1), r is the perturbation obtained by DLFuzz for x.

C. Guided Adversarial Input Search Algorithm

The guided adversarial search is the main component of DLFuzz. It is completed by solving a joint optimization problem of both maximizing the neuron coverage and the number

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2020.2997359, IEEE Transactions on Network Science and Engineering

of incorrect behaviors. The core process of the mutation algorithm is presented in Fig. 2. The algorithm contains three key components to discuss in detail.

Input: input_list ← unlabeled inputs for testing
dnn ← DNN under test
k ← top k labels different from the original label
m ← number of neurons to cover
λ ← hyperparameter for balancing the two goals
strategies ← strategies for neuron selection
cov_tracker ← tracks the information of neurons
iter_times ← iteration times for each seed
Output: set of adversarial inputs, neuron coverage
1: adversarial_set = []
2: **for** i = 0 to len(input_list) **do**
3: x = input_list[i] *//the original input*
4: seed_list = [x] *//seeds for each input*
5: **for** x_s in seed_list **do**
6: c, c_topk = dnn.predict(x_s)
7: neurons = selection(dnn, cov_tracker, strategies, m)
8: obj = sum(c_topk) - c + λ · sum(neurons)
9: grads = $\partial obj/\partial x_s$ *//gradient obtained*
10: **for** iter = 0 to iter_times **do**
11: /**grads processed to get the perturbation for mutation*/*
12: perturbation = processing(grads)
13: x' = x_s + perturbation *//mutated input obtained*
14: c' = dnn.predict(x') *//label after mutation*
15: update cov_tracker *//update coverage information*
16: $l_2\_distance$ = distance(perturbation, x)
17: **if** the coverage improved by $x' > 0.01/(i+1)$ and $l_2\_distance < 0.02$ **then**
18: seed_list.append(x')
19: **if** c' != c **then**
20: adversarial_set.append(x')
21: **break**

Fig. 2: The overall process of the mutation algorithm

Objective definition. As discussed in Section I, the gradient-based adversarial deep learning outperforms the other approaches in several aspects, especially in time efficiency. It finds perturbations by optimizing the input to maximize the prediction error [5], which is opposite to optimizing the weights to minimize the prediction error while training the DNN. It is easy to implement by customizing the loss function as our objective and maximizing the loss by gradient ascent. The loss function of DLFuzz is defined as the following equation (line 8 in Fig. 2), which is also the optimization objective:

$$obj = \sum_{i=0}^{k} c_i - c + \lambda \cdot \sum_{i=0}^{m} n_i \tag{3}$$

where the objective consists of two parts. In the first part $\sum_{i=0}^{k} c_i - c$, c is the original class label of the input, $c_i (i = 0, ..., k)$ is one of the top k class labels with confidence just lower than c (line 6 in Fig. 2). Maximizing the first part guides the input to cross the decision boundary of the original class and lie in the decision space of top k other classes. Such modified inputs are more likely to be classified incorrectly. In the second part $\sum_{i=0}^{m} n_i$, n_i is a target neuron intended to activate. These neurons are selected considering many strategies to improve the neuron coverage (line 7 in Fig. 2). The hyperparameter λ is used for balancing the two objectives.

Adversarial input search. The adversarial search reveals the overall workflow of Fig. 2. When given a test input x, DLFuzz maintains a seed list for keeping the intermediate mutated inputs which contribute to the neuron coverage. At first, the seed list only has one input which is exactly x. Next, DLFuzz traverses each seed x_s and obtains the elements making up its optimization objective. Then, DLFuzz computes the gradient direction for later mutation. In the mutation process, DLFuzz iteratively applies the processed gradient as the perturbation to x_s and obtains the intermediate input x'. After each mutation, the intermediate class label c', coverage information, l_2 distance of x and x' are acquired. If the neuron coverage improved by x' and the l_2 distance are desired, x' will be added into the seed list. Finally, if c' is already different from c, the mutation process for seed x_s terminates and x' will be included in the set of adversarial inputs. Therefore, DLFuzz is able to generate multiple adversarial inputs for a certain original input and explore a new way to further improve the neuron coverage.

For the iterative mutation process, it contains two steps. First, various processing methods are available to generate perturbations when the gradients obtained, including just keeping the sign [5], imitating the realistic situations [10], [36], etc. These mutation strategies for the input are easy to be extended to DLFuzz. Second, DLFuzz adopts l_2 distance to measure the perturbation so as to ensure the distance between x and x' is imperceptible. As for the conditions of seed keeping in line 17, DLFuzz limits our desired distance to a relatively small range (less than 0.02) to ensure the imperceptibility. As the neuron coverage improvement of one input declines with more inputs tested, the corresponding threshold for keeping the seeds also decreases with the number of inputs tested. Furthermore, we can increase the thresholds of seed keeping to reserve more mutated inputs with greater distance.

Strategies for neuron selection. To maximize the neuron coverage, we propose four heuristic strategies for selecting neurons more likely to improve coverage based on our premier work [14], as below:

1) Strategy 1. Select neurons covered frequently during past testing. Inspired by the practical experience in traditional software testing that code fragments often or rarely executed are more possible to introduce defects. Neurons covered often or rarely perhaps can result in unusual logics and activate more neurons.
2) Strategy 2. Select neurons covered rarely during past testing due to the considerations stated above.
3) Strategy 3. Select neurons with top weights. It is presented based on our assumption that neurons with top weights may have a larger influence on other neurons.
4) Strategy 4. Select neurons near the activation threshold. It is easier to accelerate if activating/deactivating neurons with output value slightly smaller/larger than the threshold.

For each seed x_s, m neurons will be selected utilizing one or more strategies, which can be customized in $strategies$ of the algorithm inputs of Fig. 2.

IV. Experiment

A. Experiment Setup

Implementation. We implemented DLFuzz based on various widespread frameworks of deep learning systems, Tensorflow 1.2.1, Keras 2.1.3 and Caffe 1.0.0. DLFuzz exhibits high portability on these general frameworks. We developed and evaluated DLFuzz on a computer with 4 cores (Intel i7-7700HQ @3.6GHz), 16GB of memory, an NVIDIA GTX 1070 GPU and Ubuntu 16.04.4 as the host OS.

Models and data sets. DLFuzz mainly tests CNN models of classification tasks, similar to mainstream adversarial testing frameworks. For evaluation, we selected two datasets used by DeepXplore for image classification tasks, MNIST and ImageNet. The same as DeepXplore, DLFuzz tested the three identical pre-trained CNNs for each dataset.

MNIST [15]: a large database of handwritten digits consisting of 60000 training images and 10000 testing images. The three pre-trained neural networks are the open-source CNNs constructed by DeepXplore based on the LeNet family [37], namely, LeNet-1, LeNet-4, LeNet-5.

ImageNet [16]: a large visual database containing over 14 million images for object recognition. The three pre-trained CNNs to evaluate are VGG-16 [35], VGG-19 [35], ResNet50 [2], which are well-known for their remarkable performance.

Considering the fairness, we also randomly choose 20 images from the dataset for each CNN as test inputs in the same way with DeepXplore. If not specified, the default settings of hyperparameters k, m, λ, $strategies$ and neuron activation threshold t are $4, 10, 1$, "$Strategy\ 1$" and 0.25 respectively. Hyperparameter $iter\_times$ is 3 for ImageNet and 5 for MNIST. The results of DeepXplore are obtained under its recommended setting.

Research questions. We constructed the experiments to answer the following three research questions to demonstrate the efficiency of our approach.

1) **RQ1.** How is the effectiveness of DLFuzz approach? (Section IV-B)
2) **RQ2.** How is the effectiveness of strategies for neuron selection? (Section IV-C)
3) **RQ3.** Is DLFuzz helpful for improving the deep learning systems? (Section IV-D)

B. RQ1: Effectiveness of DLFuzz.

Table I presents the effectiveness of DLFuzz compared with DeepXplore. DLFuzz exhibits its advantages in improving the neuron coverage, generating more adversarial inputs within the same time limit and restricting imperceptible perturbations.

Higher neuron coverage. As presented in the fifth column of Table I, for the tested six CNNs, DLFuzz achieved 1.10% to 5.59% higher neuron coverage than DeepXplore under different settings on average, including different neuron selection strategies applied and different activation thresholds for computing the neuron coverage. For the best setting, DLFuzz is able to acquire 13.42% higher neuron coverage.

As for the improvement of neuron coverage by DLFuzz, the main reasons lie in two aspects. First, DLFuzz selects more neurons with various strategies to cover specific decision logics, whereas DeepXplore randomly selects one neuron to cover, which is inadequate to improve the coverage. Next, DLFuzz maintains a seed list to keep the intermediate mutated inputs which could increase the neuron coverage during testing, which are used for subsequent testing to further improve the coverage.

More adversarial inputs. DLFuzz averagely generated 338.59% more adversarial inputs, which could be extracted from the eighth column of Table I. Several samples of adversarial inputs obtained by DeepXplore and DLFuzz for the same inputs are given in Fig. 3. Note that, DLFuzz is effective to produce adversarial inputs for any class, no matter how many classes the tested model has. Because, it is easy for DLFuzz to guide the model to classify inputs as other classes, with no relevance to number of classes.

Since DeepXplore applies three types of domain-specific constraints to the input image, namely different intensities of lights, occlusion by a single small rectangle and occlusion by multiple tiny black rectangles, one sample for each type of the constraints is listed for MNIST except the third type for ImageNet (not acquired after several times of testing). The third sample for ImageNet is substituted by another sample of the second type.

However, adversarial inputs under the third constraint for MNIST are easily acquired. Similar to the third sample of DeepXplore for MNIST, there are a few cases that the tiny black rectangles applied overlap the key features of the inputs and more tiny rectangles could even cover one part of the inputs and change their labels. This type of realistic transformations is only applicable to cross-referencing frameworks and hard to ensure the identity after being applied to the input. Although the adversarial inputs should be acquired after more experiments for ImageNet, it can be inferred that it's harder to generate adversarial inputs for larger-size colorful images than smaller-size grayscale images by putting on multiple tiny black rectangles.

Moreover, like the samples of DeepXplore in the first row of Fig. 3, lighting effects added to the inputs to obtain adversarial inputs are always stronger than in realistic conditions. In the set of adversarial inputs generated by DeepXplore under the first constraint, 60% and 25% of them are totally black or white for ImageNet and MNIST respectively. In our opinions, realistic transformations are valuable but more exploration is needed for transformations like the second constraint, which are always realistic and appropriate to obtain perturbations.

Smaller perturbations. Adversarial inputs generated by DLFuzz have 89.82% smaller perturbations, derived from the eleventh column of Table I. In this way, DLFuzz provides a stronger guarantee for the consistence of the image's identity before and after mutation. As the samples in Fig. 3, the perturbations generated by DeepXplore are visible, whereas those generated by DLFuzz are imperceptible and require careful observations to identify. Thus, the visualized perturbation is given for each sample of DLFuzz. As samples for MNIST are nearly 8 times smaller than samples for ImageNet, the alignment in Fig. 3 zooms in the samples for MNIST and magnifies their perturbations.

Higher time efficiency. Finally, DLFuzz spent 20.11% shorter time on generating each adversarial input on these

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2020.2997359, IEEE Transactions on Network Science and Engineering

TABLE I: Effectiveness of DLFuzz compared with DeepXplore, presented with the results of DeepXplore (DX.), DLFuzz (DF.) and the improvement of DLFuzz w.r.t DeepXplore.

DataSet	CNN	NC.[1]			# Adv.[2]			$\overline{l_2\ dist.}$			Time per adv.[3]		
		DX.	DF.	Improved	DX.	DF.	Increased	DX.	DF.	Decreased	DX.	DF.	Saved
MNIST	LeNet-1	51.45%	53.90%	2.45%	20	53	165.00%	8.2637	0.2708	96.72%	0.7078	0.5623	20.56%
	LeNet-4	61.50%	67.09%	5.59%	20	47	135.00%	8.2637	0.2812	96.60%	0.7078	0.6344	10.37%
	LeNet-5	63.30%	65.53%	2.23%	20	54	170.00%	8.2637	0.267	96.77%	0.7078	0.587	17.07%
ImageNet	VGG16	39.68%	43.19%	3.52%	13	89	584.62%	0.0817	0.0167	79.56%	10.473	3.4537	67.02%
	VGG19	38.43%	40.71%	2.28%	13	81	523.08%	0.0817	0.0154	81.15%	10.473	3.6606	65.05%
	ResNet50	56.00%	57.10%	1.10%	13	72	453.85%	0.0817	0.010	88.13%	10.473	16.6958	-59.42%

[1] Neuron coverage. Achieving higher neuron coverage means that it's able to test more logics of the deep learning system.
[2] The number of the adversarial inputs. More adversarial inputs could augment the training set and further be used for improving the accuracy of the model.
[3] Time for each adversarial input. Shorter time used indicates the higher efficiency.

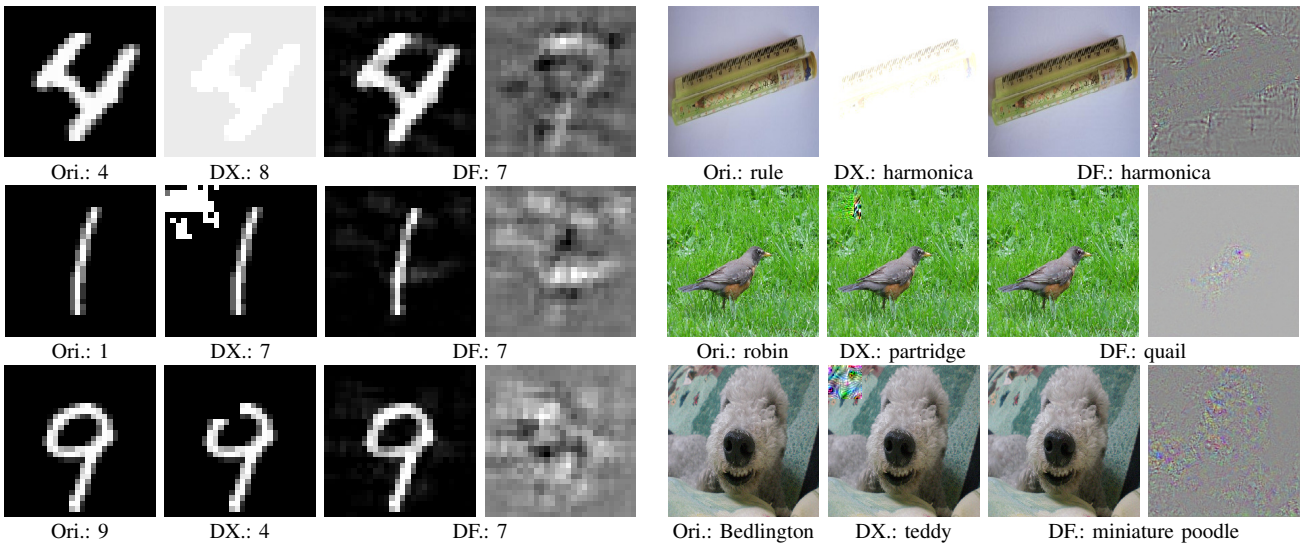


Fig. 3: Samples of adversarial inputs annotated with the framework and the predicted label. (Ori. is the original label. Each sample of DLFuzz also follows with its corresponding perturbation). The left four columns for MNIST and the right four columns for ImageNet.

deep learning systems, computed from the last column of Table I. An exceptional case is that DLFuzz cost more time in generating adversarial inputs than DeepXplore for ResNet50, which is owing to more time needed for neuron selection when testing a deep learning system consisting of a huge number of neurons (94056).

Moreover, as in twelfth and thirteenth columns in Table I, generating per adversarial input for models of ImageNet costs more time than models of MNIST. This is due to higher complexity of models of ImageNet. But, DLFuzz could still reduce the time spent on VGG models around 65%.

Neuron coverage with different activation thresholds. As illustrated in equation (1), the value of neuron coverage varies with the activation threshold customized. Fig. 4 presents the neuron coverage of DeepXplore and DLFuzz with different activation thresholds for the two datasets. DLFuzz averagely achieved 3.42% and 3.13% higher neuron coverage than DeepXplore for MNIST and ImageNet respectively under these thresholds.

Hyperparameters for configuration. As for the hyperparameters configured in the input, we tried combinations of possible settings to evaluate their influence. Table II, III, and IV reveal the variations in the effectiveness of DLFuzz with three main hyperparameters m, k, λ in equation (3), respectively. In Table III and IV, the number of neurons selected m is 8 and

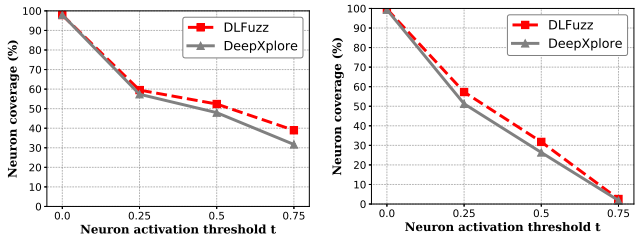


Fig. 4: The neuron coverage improvement with different activation thresholds. The four thresholds are the exact values in the evaluation of DeepXplore.

all the strategies are used together. The four main metrics are still utilized for measuring the effectiveness, different from just time efficiency evaluated in DeepXplore. For simplicity, results for several CNNs are given.

According to the experimental results, the optimal value for m is 5 among these deep learning systems and 30 is another good choice for LeNet-4 and LeNet-5. Selecting more neurons would gain no improvement in the neuron coverage and obtain fewer adversarial inputs. Here, more neurons selected will decentralize the value increasement of each neuron, as well as the weight of the part searching for adversarial inputs. Then, $k = 1$ and $k = 9$ both perform better on these CNNs and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2020.2997359, IEEE Transactions on Network Science and Engineering

$k = 3$ is also a good choice for CNNs on MNIST. $\lambda = 0.1$ is optimal for LeNet-5, VGG-16 and VGG-19 while $\lambda = 5$ is optimal for LeNet-4. But the changes of λ have little influence on the effectiveness of DLFuzz on LeNet-1 and ResNet50.

Table V provides the results of DLFuzz with changes of hyperparameter $iter\_times$ (Line 10 in Fig. 2). The parameter $iter\_times$ also has a key impact on the success rate of generating adversarial inputs given a set of test inputs. The larger $iter\_times$ is, the success rate tends to be higher, as well as the neuron coverage and the number of adversarial inputs. But more iteration times of mutating would result in larger perturbations. The optimal value for $iter\_times$ of DLFuzz is 10 for MNIST and 30 for ImageNet, whereas 50 is the default value of DeepXplore.

The answer to RQ1: DLFuzz is an effective approach for deep learning testing. It could averagely achieve 2.86% higher neuron coverage, generate 338.59% more adversarial inputs with 89.82% smaller perturbations, and save 20.11% time consumption.

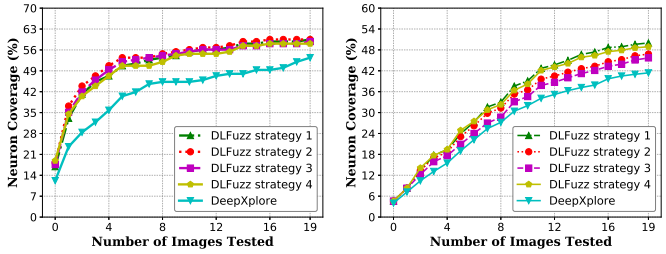
TABLE II: The variation in the effectiveness of DLFuzz on VGG-19 with different numbers of neurons selected to activate (parameter m in equation (3)). The best result across each column is highlighted in bold. (the same in the following tables).

m	NC.	# Adv.	$\overline{l_2\ dist.}$	Time per adv.
5	**45.3%**	**36**	**0.0146**	**2.8626**
8	43.7%	28	0.0147	3.7750
10	43.6%	23	0.0160	3.9238
20	42.3%	14	0.0152	7.2600
30	42.2%	12	0.0169	10.4903
40	43.4%	22	0.0177	5.8395
DX.	38.6%	15	0.1478	6.5717

TABLE III: The variation in the effectiveness of DLFuzz with different numbers of top other labels (parameter k in equation (3)).

CNN	k	NC.	# Adv.	$\overline{l_2\ dist.}$	Time per adv.
LeNet-5	1	**70.9%***	**66**	**0.2492**	**0.1989**
	3	70.5%	57	0.2592	0.2705
	4	70.5%	49	0.2683	0.3347
	9	**71.3%**	55	**0.2636***	0.4316
	DX.	69.8%	20	8.4853	0.4132
VGG-16	1	**50.3%**	**68**	0.03	**1.9574**
	3	47.2%	20	**0.0175**	2.9308
	4	48.3%	32	0.0194	2.5996
	9	**49.9%***	**63***	0.0263	2.6021
	DX.	42%	15	0.1478	6.5717

* The second largest value across the corresponding column.



LeNet-4 VGG16

Fig. 5: Neuron coverage with the number of images tested when different strategies applied in DLFuzz.

TABLE IV: The variation in the effectiveness of DLFuzz with the hyperparameter balancing the neuron coverage and adversarial inputs generating (parameter λ in equation (3)). The threshold t to compute NC. for LeNet-4 is 0.75.

CNN	λ	NC.	# Adv.	$\overline{l_2\ dist.}$	Time per adv.
LeNet-4	0.1	43.9%	42	0.2853	0.3588
	0.5	43.9%	43	**0.2838**	0.3539
	1	43.2%	**44**	0.2848	**0.3454**
	5	**45.3%**	34	0.2941	0.4226
	10	44.6%	31	0.2958	0.4663
	DX.	34.5%	20	8.5498	0.4427
VGG-19	0.1	**43.9%**	**34**	**0.0200**	**2.7917**
	0.5	43.8%	27	0.0251	3.7126
	1	42.9%	20	0.0231	4.6690
	5	42.7%	13	0.0251	6.4588
	10	42.1%	10	0.0252	7.8549
	DX.	38.6%	15	0.1478	6.5717

TABLE V: The variation in the effectiveness of DLFuzz on ResNet50 with the maximum iteration times when mutating the seeds (parameter $iter\_times$ in Line 10 of Fig. 2).

iters	success rate[1]	NC.	# Adv.	$\overline{l_2\ dist.}$	Time per adv.
3	0.6	74.3%	40	**0.0106**	8.6322
5	0.75	74.6%	75	0.0137	6.0317
10	0.9	74.9%	165	0.0190	4.8113
20	**1**	75.2%	346	0.0275	4.0486
30	**1**	75.4%	600	0.0359	**3.9829**
50	0.95	**75.6%**	**943**	0.0465	4.1807

[1] The percent of test inputs which could be mutated to be adversarial inputs.

TABLE VI: Better strategies for each CNN. The number of neurons selected m is 8. The hyperparameter λ is 0.1. The threshold t to compute NC. for LeNet models is 0.75.

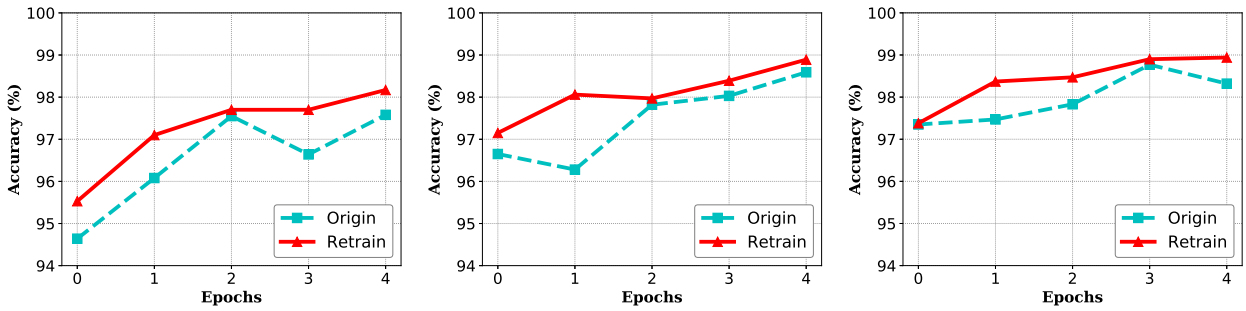
CNN	Strategies	NC.	# Adv.
VGG-16	13	50.0%	70
VGG-19	1	44.9%	43
Resnet50	2	74.5%	50
	134	74.4%	55
LeNet-1	(with Strategy 4)	36.5%	49
LeNet-4	123	43.9%	45
	23	43.9%	44
LeNet-5	1234	36.6%	50
	124	36.6%	51

C. RQ2: Effectiveness of Strategies for Neuron Selection.

We tried the four proposed strategies for neuron selection on two CNNs and depicted the results in Fig. 5. All strategies are shown to contribute more to the neuron coverage improvement than DeepXplore while having similar performance among themselves. It seems that strategy 1 performs slightly better. Besides, DLFuzz is able to achieve higher neuron coverage in the early stage.

These strategies can also be combined together and Table VI lists the better strategies for each CNN with NC. and # Adv. as metrics, since the other metrics vary little among these strategies. We have experimented over all the combinations of the four strategies on each model. For instance, "strategy 13" represents that strategy 1 and 3 are combined to select neurons to cover, the same with other cases. For LeNet-1, combinations with strategy 4 all perform well. As shown in Table VI, the better strategies for each CNN are not common, so various strategies designed provide the higher possibility for maximizing the neuron coverage.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2020.2997359, IEEE Transactions on Network Science and Engineering



LeNet-1 LeNet-4 LeNet-5

Fig. 6: The accuracy improvement of three CNNs on MNIST after retraining.

The answer to RQ2: The neuron selection strategies we designed are effective. All the neuron selection strategies perform better than DeepXplore while the better strategies for each deep learning system vary.

D. RQ3: Improving deep learning Systems by DLFuzz.

To prove the practical use of DLFuzz, we incorporated 114 adversarial images into the training set of three CNNs on MNIST and retrained them to see if their accuracy is able to increase. Note that it is hard to improve their accuracy as their accuracy is already around 98% and those adversarial images take up a tiny part with respect to the whole 60000 training images. Finally, we improved their accuracy by up to 1.8% within 5 epochs. More improvement is expected if more adversarial inputs included in the retraining process. In this way, these adversarial inputs generated by DLFuzz augmented the training dataset with more corner cases and could improve the robustness of the deep learning systems.

The answer to RQ3: DLFuzz exhibits its practical use for improving the deep learning systems. By augmenting the training dataset and retraining the deep learning systems, DLFuzz could improve their accuracy by up to 1.8% within 5 epochs.

E. Discussion

Applicability of fuzzing to deep learning testing. The effectiveness of DLFuzz demonstrates that applying the knowledge of fuzzing to deep learning testing is feasible and can greatly improve the performance of existing deep learning testing techniques such as DeepXplore. The gradient-based solution of the optimization problem guarantees the easy deployment and high efficiency of the framework. The mechanism of seed maintenance provides diverse directions and larger space for improving the neuron coverage. Besides, DLFuzz is capable to obtain incremental adversarial inputs for one input. Various strategies combined for neuron selection proved to be good at finding neurons beneficial for increasing the neuron coverage.

Without manual effort. For confirmation, we checked all the 366 adversarial inputs generated by DLFuzz, though DLFuzz maintains quite small l_2 distance by the restricted threshold. We haven't found any adversarial inputs that have already changed their identities after mutation. The adversarial inputs are nearly the same as the original input, and the perturbations are imperceptible.

Advances over networking systems. Although DLFuzz currently aims at image classification tasks, which are almost CNN models involved, DLFuzz is designed with scalability to be applied to other tasks and systems. For the advances of our approach to be used over networking systems, they could be summarized as twofold. First, CNN is now widely used in many networking systems, always as necessary components playing artificial tasks, like video processing component in large-scale video storage networking system. Second, CNN is also a special networking system, depending on connected layers and neurons to process huge information. Thus, the advances of our method over CNN are potential to be generalized to common networking systems.

Future work. Encouraged by the impressive effects of DLFuzz on image classification tasks, we will work on the deployments of DLFuzz on other popular tasks in deep learning domains, such as speech recognition. The specific constraints for input mutation of the corresponding task will be added into the common workflow. Also, some domain knowledge can be leveraged to provide more efficient mutation operations and increase the efficiency of DLFuzz.

V. Conclusion

We design and implement DLFuzz as an effective coverage guided adversarial testing framework of deep learning systems. DLFuzz first combines the basic ideas of fuzz testing into deep learning testing and demonstrates its effectiveness. Compared with DeepXplore, DLFuzz averagely obtained 2.86% higher neuron coverage and generated 338.59% more adversarial inputs with 89.82% smaller perturbations given the same amount of inputs. DLFuzz also overcomes the trouble of relying on multiple deep learning systems of the similar functionality in DeepXplore. The novel strategies designed by DLFuzz for neuron selection perform well in improving the neuron coverage. Additionally, DLFuzz exhibits its practical use by incorporating these adversarial inputs to retrain the deep learning systems and to steadily improve their accuracy.

Currently, DLFuzz is evaluated for CNN models and image classification tasks. Since the scalability of its design, our future work mainly includes the generalization over RNN models and corresponding domains, like speech recognition and natural language processing. Finally, our framework will try to provide solutions for common networking systems.

ACKNOWLEDGMENT

This research is sponsored in part by National Key Research and Development Project (Grant No. 2019YFB1706200), the NSFC Program (No. U1911401, 61802223), the Huawei-Tsinghua Trustworthy Research Project (No. 20192000794), and the Equipment Pre-research Project (No. 61400010107).

REFERENCES

[1] 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.

[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] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[4] M. Helmstaedter, K. L. Briggman, S. C. Turaga, V. Jain, H. S. Seung, and W. Denk, "Connectomic reconstruction of the inner plexiform layer in the mouse retina," *Nature*, vol. 500, no. 7461, p. 168, 2013.

[5] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *Proceedings of the 2nd International Conference on Learning Representations*, 2014.

[6] S. M. Moosavi Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," in *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, no. EPFL-CONF-218057, 2016.

[7] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *International Conference on Computer Aided Verification*. Springer, 2017, pp. 3–29.

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

[9] M. Wicker, X. Huang, and M. Kwiatkowska, "Feature-guided black-box safety testing of deep neural networks," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2018, pp. 408–426.

[10] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 1–18.

[11] M. Zalewski, "American fuzzy lop," 2007.

[12] M. Böhme, V.-T. Pham, and A. Roychoudhury, "Coverage-based greybox fuzzing as markov chain," *IEEE Transactions on Software Engineering*, 2017.

[13] M. Böhme, V.-T. Pham, M.-D. Nguyen, and A. Roychoudhury, "Directed greybox fuzzing," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 2329–2344.

[14] J. Guo, Y. Jiang, Y. Zhao, Q. Chen, and J. Sun, "Dlfuzz: differential fuzzing testing of deep learning systems," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2018, pp. 739–743.

[15] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[16] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 248–255.

[17] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *Computer Science*, 2015.

[18] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE, 2016, pp. 372–387.

[19] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *Proceedings of the 2nd International Conference on Learning Representations*, 2017.

[20] Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, and J. Li, "Boosting adversarial attacks with momentum," in *Proceedings of 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, Spotlight.

[21] J. Su, D. V. Vargas, and S. Kouichi, "One pixel attack for fooling deep neural networks," *arXiv preprint arXiv:1710.08864*, 2017.

[22] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 38th IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 39–57.

[23] J. Guo, Y. Zhao, X. Han, Y. Jiang, and J. Sun, "Rnn-test: Adversarial testing framework for recurrent neural network systems," *arXiv preprint arXiv:1911.06155*, 2019.

[24] L. Wu, X. Du, W. Wang, and B. Lin, "An out-of-band authentication scheme for internet of things using blockchain technology," in *2018 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2018, pp. 769–773.

[25] N. Akhtar and A. Mian, "Threat of adversarial attacks on deep learning in computer vision: A survey," *IEEE Access*, vol. 6, pp. 14 410–14 430, 2018.

[26] L. Pulina and A. Tacchella, "An abstraction-refinement approach to verification of artificial neural networks," in *International Conference on Computer Aided Verification*. Springer, 2010, pp. 243–257.

[27] X. Huang and X. Du, "Achieving big data privacy via hybrid cloud," in *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2014, pp. 512–517.

[28] L. Ma, F. Juefei-Xu, J. Sun, C. Chen, T. Su, F. Zhang, M. Xue, B. Li, L. Li, Y. Liu *et al.*, "Deepgauge: Comprehensive and multi-granularity testing criteria for gauging the robustness of deep learning systems," *The 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2018.

[29] (2017, Aug.) libfuzzer in chrome. [Online]. Available: https://llvm.org/docs/LibFuzzer.html

[30] M. Wang, J. Liang, Y. Chen, Y. Jiang, X. Jiao, H. Liu, X. Zhao, and J. Sun, "Safl: increasing and accelerating testing coverage with symbolic execution and guided fuzzing," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. ACM, 2018, pp. 61–64.

[31] O. Bastani, R. Sharma, A. Aiken, and P. Liang, "Synthesizing program input grammars," in *ACM SIGPLAN Notices*, vol. 52, no. 6. ACM, 2017, pp. 95–110.

[32] J. Wang, B. Chen, L. Wei, and Y. Liu, "Skyfire: Data-driven seed generation for fuzzing," in *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 2017, pp. 579–594.

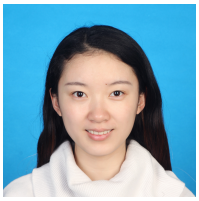
[33] D. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. G. Kang, Z. Liang, J. Newsome, P. Poosankam, and P. Saxena, "Bitblaze: A new approach to computer security via binary analysis," in *International Conference on Information Systems Security*. Springer, 2008, pp. 1–25.

[34] N. Stephens, J. Grosen, C. Salls, A. Dutcher, R. Wang, J. Corbetta, Y. Shoshitaishvili, C. Kruegel, and G. Vigna, "Driller: Augmenting fuzzing through selective symbolic execution." in *NDSS*, vol. 16, 2016, pp. 1–16.

[35] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proceedings of the International Conference on Learning Representations*, 2015.

[36] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th International Conference on Software Engineering*. ACM, 2018, pp. 303–314.

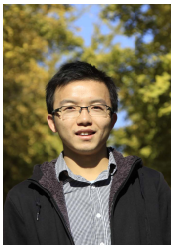
[37] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.



Jianmin Guo is a Ph.D. candidate at School of Software Engineering, Tsinghua University, Beijing, China. She received the BS degree in software engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 2017. Her research interests are software testing, mainly focusing on deep learning testing and backdoor detection of deep learning systems.

Houbing Song received the Ph.D. degree in electrical engineering from the University of Virginia, Charlottesville, VA, in 2012, and the M.S. degree in civil engineering from the University of Texas, El Paso, TX, in 2006. In 2017, he joined the Department of Electrical Engineering and Computer Science, Embry-Riddle Aeronautical University, Daytona Beach, FL, where he is currently an Assistant Professor and the Director of the Security and Optimization for Networked Globe Laboratory. His research interests include cyber-physical systems, cybersecurity and privacy, internet of things, edge computing, AI/machine learning, big data analytics, unmanned aircraft systems, connected vehicle, smart and connected health, and wireless communications and networking.

Yue Zhao is a master student at School of Software Engineering, Tsinghua University, Beijing, China. He received the BS degree in software engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 2017. His research interests are deep learning testing and backdoor detection of deep learning systems.



Yu Jiang received the BS degree in software engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 2010, and the PhD degree in computer science from Tsinghua University, Beijing, China, in 2015. He was a Postdoc researcher in the department of computer science of University of Illinois at Urbana-Champaign, IL, USA, in 2016. He is now an associate professor in Tsinghua University, Beijing, China. His current research interests include domain specific modeling, formal computation model, formal verification and their applications in embedded systems.