

FuzzGAN: A Generation-Based Fuzzing Framework For Testing Deep Neural Networks

Abstract—Deep neural networks (DNNs) are increasingly deployed in a wide variety of fields. Despite their spectacular advances, DNNs are known to suffer from various vulnerabilities. Similar to traditional software, one of the most common DNN vulnerabilities relates to the hidden defects that allow unexpected inputs (adversarial examples) lead a DNN to incorrect classifications. Fuzzing technique has been frequently used to evaluate traditional software and it is also a way to effectively discover potential defects hidden within a DNN. In this paper, we propose a generation-based fuzzing framework FuzzGAN to detect the defects existing in DNNs. Rather than the mutation-based fuzzing that tests DNNs with mutated seed inputs, FuzzGAN generates test cases from random noise under the guidance of some coverage criterion, without the limitation of concrete seeds. We take neuron coverage as our criterion to measure the adequacy of the fuzzing and investigate the effectiveness of FuzzGAN on two DNN models that are with classical network structures and trained on public datasets. The experiment demonstrates that FuzzGAN can generate realistic and valid test cases and achieve a high neuron coverage. Moreover, these test cases can be used to improve the performance of the target DNN through adversarial retraining.

Index Terms—deep neural networks, fuzz testing, neuron coverage

I. INTRODUCTION

Deep neural networks (DNNs) are a set of machine learning algorithms modeled loosely after the biological neural networks to progressively approach their tasks based on data representation learning [20]. Over the past few years, DNNs, whose performance is comparable to and in some cases superior to human contestants, have made impressive progress. The widespread adoption makes it crucial to discuss security properties of DNNs, one of which has attracted wide attention is the robustness against adversarial examples.

For traditional software, the commonly used approach of discovering potential defects and evaluating security properties is fuzzing, which exercises the tested program by a series of synthetic inputs (i.e. test cases) and observes corresponding resultant behaviors to determine whether this program satisfies particular security requirements. There are two primary components in fuzzing: test coverage criterion and test case generation algorithm [6].

According to the characteristics of the program to be tested, there are diverse coverage criteria proposed for fuzzing. The most commonly used criteria for traditional software (e.g. a program shown Fig. 1 (left)), such as code coverage, path coverage and so on, usually estimate the amount of the code exercised by a series of test cases (i.e. a test suite). However, a program based on DNN (Fig. 1 (right)) is usually simple and

straightforward, such that being exercised by any single test case can easily reach a fairly high code coverage. Actually, a DNN always has complex network structure and plenty of links and parameters. As shown in Fig. 1 (right), its mapping from input to output is learned from abundant training data rather than any objective-specific programming and is represented in network structure and inner weights rather than the program that invokes it. Hence, for applying fuzzing techniques to the evaluation of DNNs, new test coverage criteria have been specifically proposed, e.g. neuron coverage [17], neuron bound coverage [10], sign-sign coverage [23], t-way combination sparse coverage [11], etc.

According to the manner in which a fuzzer generates test cases, current fuzzing techniques can be divided into mutation-based fuzzing that generate test cases by modifying existing inputs, and generation-based fuzzing that generates from scratch [5]. Compared with mutation-based fuzzing, generation-based fuzzing requires more knowledge on the target program and is harder to start, while its generated test cases can more strongly pass the input-validation and achieve a higher coverage. Existing work on fuzz testing of DNNs [10], [11], [14], [17], [23], [26] are all mutation-based fuzzing that generates test cases by modifying given inputs (i.e. seeds).

In order to fill up the blank in generation-based DNN fuzzing, we focus on the proximity of generated test cases to real data and propose a fuzzing framework FuzzGAN that generates realistic and valid test cases from random noise. With plenty variants of GAN [2], [19], [28], [29], adversarial training technique has achieved satisfying performance on generative tasks. Therefore, we take advantage of GAN to learn the representation of real data, so that the generated inputs locate in the input space of the tested DNN. Moreover, we utilize the tested DNN as an oracle, and linearly combine GANs fundamental target of generating indistinguishable input samples with the goals triggering incorrect behaviors and enhancing test coverage. Under the combined objectives, we train a generator against a discriminating network and employ it to generate test cases for maximizing neuron coverage during the fuzz testing.

The main contributions of this work include:

- We propose a generation-based fuzzing framework FuzzGAN for evaluating the robustness of DNNs against adversarial examples. Under the guidance of coverage criterion, the test case generator is trained by delicately tailored adversarial networks, so that FuzzGAN can 1) generate test cases from random noise, 2) generate test

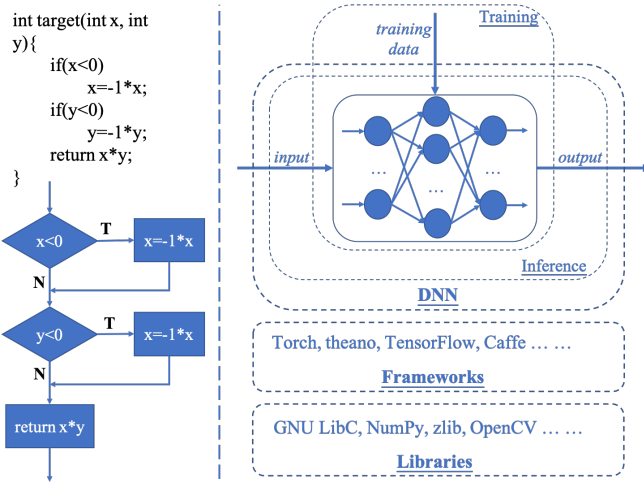


Fig. 1. The comparison between traditional software and DNNs.

cases that are indistinguishable from real inputs, 3) achieve high test coverage by a test suite.

- We have implemented FuzzGAN to separately test a LeNet-1 model trained on a handwritten digit data set MNIST and a VGG-11 model on data set LSUN. Experiments and evaluation present that FuzzGAN can effectively generate test cases and test DNN robustness with a high neuron coverage.

To the best of our knowledge, FuzzGAN is the first generation-based fuzzing framework for deep neural networks.

II. PROBLEM SCOPE AND DESIGN OBJECTIVES

A. Problem Scope and Terminology

This paper aims to test the robustness of deep neural networks against adversarial examples in white-box scenario. An **adversarial example** is a well-designed input sample to a neural network that leads to an erroneous behavior [24]. In the context of machine learning, an **erroneous behavior** is generally defined as incorrect classifications that the neural network takes an input sample and outputs a specific category or any category other than the true one. Accordingly, adversarial attacks can be divided into **targeted attacks** and **non-targeted attacks**. Depending on the capability of tester, **white-box testing** refers to a method of testing DNNs that requires the tester to have full access to the tested DNN, including training algorithm, training data, network structure, internal weights and so on.

In this paper we employ **fuzzing**, a software testing technique, to test DNNs, which provides malicious data as inputs for a DNN and monitors incorrect outputs. Fuzz testing usually consists of test coverage criteria and the algorithms of test case generation. **Test coverage criteria** measure the comprehensiveness and adequacy of the test, and guide test case generation. In this paper, FuzzGAN is guided by neuron coverage. **Neuron coverage** is defined as the ratio of the number of the neurons activated by any test case in a test

suite to the total number of neurons in the DNN under test [17].

A **neuron** is considered to be **activated** if the value of its output is high enough (higher than a threshold), such that it has impact on the output of the neurons in subsequent layers and even the output layer. Since there exist both positive values and negative values among the outputs of the neurons in a layer and the weights in subsequent layers, we regard that a larger absolute value of neuron output implies larger impact to the following layers and a zero implies no impact. Additionally considering that the neurons in a DNN may be multidimensional and the range of neuron output is dynamic, FuzzGAN takes the averaged value of the absolute values of all elements inside a neuron as its **output** value and this value is normalized, so that the neurons with various dimensions and value ranges can be compared with the same threshold.

B. Design Objectives

In this paper, we aim to propose an effective fuzzing framework that, given a specified test coverage criteria, can generate useful inputs towards the tested DNN. Specifically, the test case generation is required to satisfy the following three objective:

a) *Generating valid and indistinguishable inputs:* For testing traditional software, a fuzzing tester sometimes provides invalid, unintended, or random data as inputs for the target program and observes exceptions, like crash and potential memory overflows. However, fuzz testing that evaluate the robustness of DNN against adversarial examples monitors the incorrect classification predictions only, so that the test cases sent towards target DNN have to be valid for DNN to produce a classification output. It requires the test cases to be in the same format and in the same data space as natural input samples. Hence, the first objective of FuzzGAN is to learn the distribution of natural input and generate input samples that are indistinguishable from natural ones by human or a discriminator model.

b) *Leading to errors:* A tester usually intends to input well-designed adversarial examples as test cases to the tested DNN, so that it can detect incorrect classification predictions and reveal adversarial vulnerabilities in the DNN. The more adversarial examples among all generated inputs suggests higher efficiency of the test. Therefore, one of the objectives for designing FuzzGAN is to generate error-prone input samples, i.e. to increase the incidence of adversarial examples.

c) *Increasing test coverage:* In this paper, we take neuron coverage as test coverage criterion to guide the generation of test cases. We aim at generating test cases that can maximize the normalized absolute value of the output of specified neurons, so that we may activate any neuron and then achieve a higher neuron coverage with less number of test cases.

III. METHODOLOGY

In this section, we elaborate our generation-based DNN fuzzing framework under the guidance of neuron coverage. We give an presentation of the overall fuzzing process at first, and then describe the design of test case generator in details.

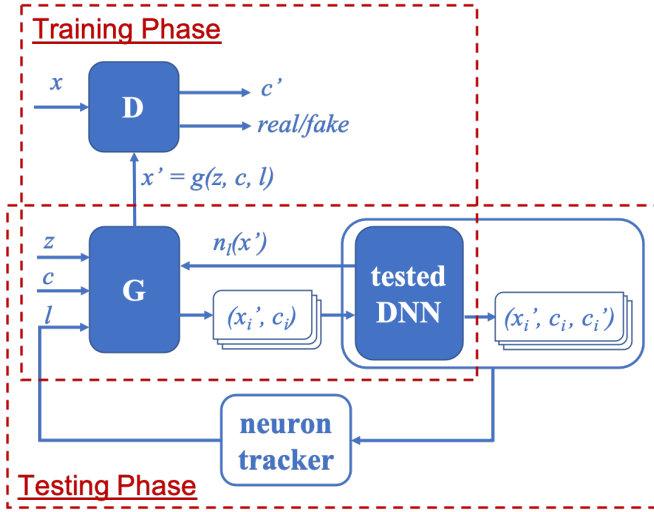


Fig. 2. An overview of the generation-based DNN fuzzing framework FuzzGAN.

A. The overview of FuzzGAN

The lifecycle of FuzzGAN contains two phases, training and testing (as shown in Fig. 2). In the training phase, a generative model that is designed as a convolutional neural network is deliberately trained through adversarial techniques to generate input samples satisfying the objectives listed in Section II-B, which will be specifically described in the next subsection III-B. This phase can be regarded as a preparation for the subsequent testing phase, in which a fuzz testing is conducted with the trained generative model as test case generator.

The overall process of testing DNN robustness with FuzzGAN is a loop that iteratively generates input samples towards the DNN under test, as shown in Algorithm 1 and Fig. 2 (Testing Phase). There is a tracker *neuron_tracker* recording activation state of all neurons in tested DNN, where a neuron is marked as "activated" if it has, so far, been activated by at least one test case in the test suite. The test case generator *G* takes three input parameters (random noise z , randomly selected class c and the location information l of a neuron n_l randomly selected among non-activated neurons in *neuron_tracker*) and outputs a synthesized input sample $x = g(z, c, l)$ to the tested DNN *dnn*.

Only the generated input samples that make the tested DNN fall into erroneous classifications ($dnn(x) \neq c$) are effective test cases for fuzzing and will be appended into the test suite *test_suite*. Once the test suite is enlarged, the neuron activation tracker *neuron_tracker* attempts to update itself by marking the neurons that have not been activated by any of previous test cases but activated by x (If $n_l(x') > t$, where t is the threshold of activation, the neuron n_l is considered as activated by x).

This loop of generating input samples will not terminate until the desired neuron coverage d is achieved. In the end, the tester obtains a suite of test cases *test_suite* through FuzzGAN, and by which the test may achieve a rather high neuron coverage.

Algorithm 1 Testing Phase of FuzzGAN

Input: $dnn \leftarrow$ the DNN under test;

$g \leftarrow$ a test case generator trained through GAN;

$C \leftarrow$ the set of all classes;

$t \leftarrow$ the threshold of activation;

$d \leftarrow$ the desired neuron coverage.

Output: *test_suite* \leftarrow a set of test cases.

```

1: function COVERAGE(neuron_tracker)
2:    $a\_num \leftarrow$  the number of activated neurons
3:    $num \leftarrow$  the number of all neurons in tested DNN
4:   return  $a\_num \setminus num$ 
5: end function
6: test_suite  $\leftarrow$  an empty set
7: neuron_tracker  $\leftarrow$  set all neurons non-activated
8: for COVERAGE(neuron_tracker)  $> d$  do
9:    $c \leftarrow$  target class randomly selected from  $C$ 
10:   $z \leftarrow$  random noise
11:   $l \leftarrow$  the location information of a randomly selected
    non-activated neuron from neuron_tracker
12:   $c \leftarrow$  target class randomly selected from  $C$ 
13:   $x' \leftarrow g(z, c, l)$ 
14:  if  $dnn(x) \neq c$  then
15:    test_suite.add( $x$ )
16:    neuron_tracker.update( $x, t$ )
17:  end if
18: end for

```

B. The design of test case generator

Since FuzzGAN is proposed as a framework of generation-based fuzzing, its test case generator is expected to synthesize test cases from scratch. For this task of generation, we design the test case generator as a multi-layer perceptron with convolutional layers and employ adversarial technique to train this generative model. In this subsection we introduce how the test case generator in FuzzGAN is designed and trained in the training phase.

The process of training test case generator is displayed in Fig. 2, besides a discriminative model, we take the tested DNN as an oracle as well and iteratively train the generative model with the aid of feedback from them. Both of the generative model and discriminative model are multi-layer perceptrons with parameters θ_G and θ_D respectively and the alternate and iterative training of them is exactly a process to adjust these parameters. During training, the generator *G* takes random noise z , randomly selected class c and the location l of a randomly selected neuron n_l as input, synthesizes a input sample x' to the tested DNN *dnn*; the discriminative model *D* takes input x_{in} from both the real data set X and the samples generated by *G*, and then classifies x_{in} as c' and distinguishes whether the input x_{in} is a real sample x ($x_{in} \in X$) or a generated one x' ($x_{in} = g(z, c, l)$).

The training process of *D* and *G* is guided by the loss functions that represent the loss of each model. By decreasing the loss with gradient descent method, the model can achieve

their functionalities. The discriminator D has two main tasks, classifying the input sample x_{in} as the tested DNN, and distinguishing whether the input is synthetic or real. Its loss function L_D is thus has two parts:

$$L_D = L_{class} + L_{real} \quad (1)$$

$$L_{class} = CrossEntropyLoss(p_c, c) \quad (2)$$

$$L_{real} = E[\log P(real|x_{in} \in X)] \\ + E[\log P(fake|x_{in} \in g(z, c, l))] \quad (3)$$

We use $p_c = \{p_1, p_2, \dots, p_k\}$ to denote a vector of confidence scores representing the confidence with which D classifies x_{in} as each class, where k is the number of classes, and use the cross-entropy loss $CrossEntropyLoss(\cdot)$ between p_c and c to represent D 's loss on classifying L_{class} that increases when the predicted probability diverges from the target class. L_{real} denotes the loss of discrimination whether an input is from a real data set, where $E(\cdot)$ calculates the value of expectation.

For the generative model G , generating valid input samples for dnn requires that the generated data is realistic enough to confuse D 's discrimination functionality, and generating a sample in the specified class c requires the the generated data to coordinate D 's classifying. In this way, G is trained to maximize D 's loss of discrimination L_{real} and minimize L_{class} .

Additionally, the generated data is supposed to address the rest objectives: leading the tested DNN to erroneous behaviors and maximize the neuron coverage. The former one will be achieved by maximizing the cross-entropy loss for dnn to classify the x' to the specified class c . While the generative model is being trained rather than being used for fuzz testing, its previous generated samples are not recorded and the neuron activation state of dnn is not traced. Hence, G is not trained to maximize neuron coverage but to maximize the output of the neuron specified by the parameter l . l denotes the location information of a neuron, including the layer l_{layer} and the concrete index l_{index} inside this layer.

Taken together, the loss function of G is designed as a linear combination of three parts:

$$L_G = L_{valid} + \lambda_1 \cdot L_{error} + \lambda_2 \cdot L_{neuron} \quad (4)$$

$$L_{valid} = L_{class} - L_{real} \quad (5)$$

$$L_{error} = -CrossEntropyLoss(dnn(x), c) \quad (6)$$

$$L_{neuron} = CrossEntropyLoss(layer(x, l_{layer}), l_{index}) \quad (7)$$

$CrossEntropyLoss(\cdot)$ here is used to measure the performance of the tested DNN on classification, and is also employed to let the output of specified neuron n_l (neuron

TABLE I
TESTED DNNs.

Parameters	LeNet-1	VGG-11
Dataset	MNIST	LSUN
Number of classes	10	4
Number of layers	6	15
Number of neurons	52	4763
Reported accuracy	98.3%	—
Reproduced accuracy	98.3%	93.40%

no. l_{index}) diverge from the others in the same layer (layer no. l_{layer}), where $layer(x, l_{layer})$ represents the outputs of all neurons in the layer containing n_l . Hyper parameters λ_1 and λ_2 are the weights directing the trade-off among the importance of each individual loss term in L_G .

In the duration of the training procedure, generator G and discriminator D are trained iteratively, and within each iteration, they are trained end to end, taking advantage of gradient decent technique, to minimize respective loss. The existence of conflict between G and D 's loss functions is where the adversarial idea shows in the adversarial technique.

As a bonus of this training scheme composed by generative adversarial networks, FuzzGAN requires no prescribed human-based rule or hand-crafted feature, and each generated sample is synthesized from random noise without being limited on any specific sample.

IV. IMPLEMENTATION

As a proof-of-concept, we implement FuzzGAN for neuron coverage guided fuzzing of two deep neuron networks. All experiments were performed in Pytorch, on a workstation with Tesla K80 GPUs¹.

A. Tested DNN

The concrete detail of tested DNNs are given in Table I. The first one is a LeNet-1 [8] that contains 6 layers and 52 neurons in total. It is a character recognizer previously trained on MNIST (Modified National Institute of Standards and Technology database [9]) with an accuracy of 98.34%. The input images to LeNet-1 are in a size of 28*28.

FuzzGAN has been further implemented to test a VGG-11 network [21] on LSUN [27]. The images in LSUN are divided into 10 categories, and the VGG-11 under test was trained on four of them (towers, restaurants, restaurants, and bridges). The tested VGG-11 takes RGB images in size of 64*64 as input and contains 15 layers.

B. Test case generator

In this subsection, we introduce the details on test case generator when implementing FuzzGAN.

¹Source code will be made available.

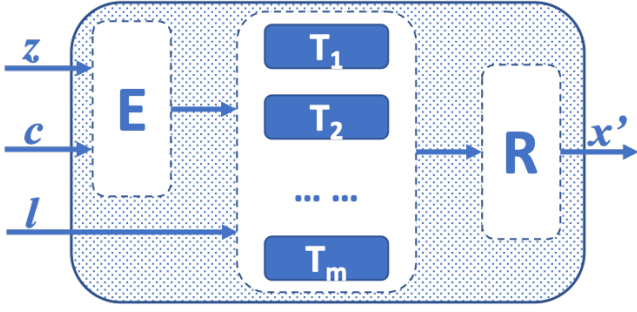


Fig. 3. The modular structure of the generator in FuzzGAN.

a) *Network architectures*: As described previously, when the test case generator is producing an input sample that enlarges the normalized absolute value of a specified neuron, it is required to take all neurons of the layer containing this specified neuron. To handle a mass of neurons in a DNN, we divide the network of generator into several reusable and composable modules (shown in Fig. 3): an encoder E encoding the noise z into a latent representation with the given class c , multiple transformers T 's transforming the output of E on the basis of the specified neuron n_l , and a reconstructor R reconstructing the output from T 's into the format acceptable for the tested DNN. The number of transformers relies on the architecture of tested DNN, i.e. the number of its layers. The transformers have the same network architecture but are trained individually for activating neurons in corresponding layers of the tested DNN.

Within FuzzGAN, the generator G and discriminator D are designed as convolutional neural networks, and the architectures of them are shown in further detail in Table II. By *Conv2d+Tanh* we denote a 2D convolutional layer with *Tanh* as activation function. By analogy, *Conv2d+BN+LR* represents a 2D convolutional layer followed by batch normalization and activation function Leaky ReLU, *DP* represents a dropout layer, and so on.

Moreover, with the development of deep learning, the DNNs have been designed to be deeper with giant number of neurons. Consequently, the architectures of networks in FuzzGAN are designed in accordance with the concrete architecture of the tested DNN. As shown in Table II, the networks for fuzz testing a VGG-11 network trained on LSUN are more complicated than those for testing a LeNet-1 network trained on MNIST.

b) *Training process*: The first step of fuzzing with FuzzGAN is to train a test case generator. All networks in FuzzGAN are initialized randomly in the beginning. During the training procedure, we adopt mini-batch stochastic gradient descent technique [2] to minimize loss functions and learn parameters for each model. The training images are fed by batch and the generator and the discriminator are trained alternatively on each batch. In the procedure of training, the expected class and the neuron location are randomly selected. To evenly train the transformers for activating the neurons in different layers, the layer in which the location of a neuron is

TABLE II
ARCHITECTURES OF NETWORKS IN FUZZGAN.

Net.	Mod.	MNIST	LSUN
G	E	Embedding Linear	Linear ResidualBlock * 2
	T	Embedding Linear	Embedding Linear ResidualBlock * 2 ResidualBlock * 8
	R	ResidualBlock * 5 ReLU Conv2d + Tanh	BN + UpSample (Conv2d + BN + LR) * 2 Conv2d + Sigmoid
D		(Conv2d + LR + DP) * 4 Linear + Sigmoid	ResidualBlock * 4 Linear

TABLE III
PARAMETER SETTINGS.

Parameters	Values	
	LeNet-1	VGG-11
ImageSize	28 * 28	64*64
TrainingSetSize	60000	2,810,853
TransformerNumber	6	15
LearningRate	0.0002	0.0004
BatchSize	128	128
Epochs	100	1200
λ_1	0.5	0.1
λ_1	diverse	diverse

sent to generator in each iteration is specified in turn.

C. Hyper parameters

The values of primary hyper parameters used in experiments are given in Table III. In the experiment, FuzzGAN is trained by traversing the dataset for Epochs times with a learning rate of LearningRate. Images from dataset were fed in batches of BatchSize, and one fed batch triggers one iteration of training steps, by which each type of modules and the discriminator were successively trained. We performed grid search to find the optimum loss weights λ_1 and λ_2 for each tested DNN. Table III shows the empirically chosen λ_1 and to enhance the output of neurons from different layers, the value of λ_2 was diverse.

V. EVALUATION

In this section, we evaluate FuzzGAN from four aspects: its respective performance on three previously mentioned objectives, and the improvement of the tested DNN by generated test cases.

A. Indistinguishability

The most basic objective of FuzzGAN is generating valid inputs that have similar distribution with natural samples from a real dataset, which implies the generated input samples are required to be indistinguishable from natural samples.

At first, we carried out an evaluation with human subjects to demonstrate the indistinguishability of test cases generated in FuzzGAN. For each of datasets, we randomly and evenly selected 10 images for each class, half of which are real

TABLE IV
NAKED-EYE DISCRIMINATION.


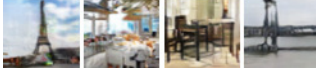
Dataset	Examples	Acc.
MNIST		49.17%
		
LSUN		57.08%

TABLE V
INCEPTION SCORE OF REAL/GENERATED IMAGES.

DataSet	IS	
MNIST	real	2.5218
	fake	2.5237
LSUN	real	6.9967
	fake	4.8107

images from original dataset (MNIST and LSUN respectively) and the other ones are generated by FuzzGAN. It can be intuitively found that the images generated by FuzzGAN look strongly alike with real ones. We invited 30 human subjects to distinguish the fake ones in each of two data sets. We display some examples of generated images in the Examples column of Table IV. For images of MNIST, our subjects achieved an averaged accuracy of 49.17% that seems to be randomly guessing. However, since images in LSUN do not have enough quantity or amount for the generator in FuzzGAN to perfectly learn the representation of complex scene, the performance of FuzzGAN on imitating LSUN is slightly weaker than imitating MNIST: subjects can distinguish the generated images from real ones with an accuracy of around 57.08%.

Furthermore, we utilize Inception Score (IS) to evaluate the quality of images generated by FuzzGAN for LeNet-1 on MNIST and VGG-11 on LSUN respectively. The inception score [18] is proposed as a metric for evaluating a generative model. It focuses on two desirable traits of generating images: each of images containing single and clear object, and generated images having high diversity. It leads to a large IS if both of these qualities are satisfied. As shown in Table V, we compare IS of images generated by FuzzGAN and that of real images from MNIST or LSUN. It shows that the quality of generated images imitating MNIST is as good as that of real ones, and the images imitating LSUN is slightly inferior to real ones.

B. Probability of leading to error

A high probability for generated input samples to make the tested DNN improperly classify always implies a high efficiency of test. To estimate the probability, we separately used FuzzGAN to generate 1000 images for LeNet-1 and VGG-11 with randomly specified classes and neuron locations. Then we sent generated images into the tested DNN and compare the label outputted by DNN and the label specified

TABLE VI
RATE OF ERRONEOUS CLASSIFICATION.

	LeNet-1	VGG-11
rate	53.5%	75.5%

TABLE VII
NEURON EXAMPLES.

Net.	Example	Data	Output		No. of activation
			Original	Processed	
LeNet-1	a	real	1.1183	0.5964	61
		fake	1.9561	0.8560	100
	b	real	-44.81	0.6130	57
		fake	-70.08	0.9853	100
VGG-11	c	real	0.0251	0.3373	0
		fake	0.0233	0.5062	35
	d	real	1.9899	0.8386	96
		fake	2.9897	0.8192	100

when generating. At last, we counted the rate of erroneous classification and showed the results in Table VI. The result proves that the generator has a high probability (more than 53.5%) of generating adversarial input samples.

C. Coverage enhancement

1) *Neuron output enhancement*: As introduced in Section III-B, the generator is designed to synthesize input samples that maximize the output of the neuron specified by its input parameter l . To evaluate this functionality, we randomly selected several neurons from each tested DNN (12 from LeNet-1 and 10 from VGG-11) and recorded their outputs when feeding images into the DNN. We separately selected 100 images from MNIST dataset and LSUN dataset as control group, and generated 100 input samples respectively for maximizing each selected neuron as experimental group. We recorded original outputs and processed outputs (normalized absolute values) of selected neurons and counted the number of images that activate the specified neuron.

We set the threshold to 0.6. Table VII displays the original and processed outputs of two randomly picked neurons from those mentioned above in each DNN as examples. With a statistical analysis, it can be observed that, for the tested LeNet-1 the processed output of selected neurons with experimental group is larger than that with control group by approximately 76.22%, and for the VGG-11, FuzzGAN can effectively increase the processed output of selected neuron by around 42.95%. Furthermore, input samples generated by FuzzGAN averagely can activate 108.90% more neurons in LeNet-1 than natural images and activate 146.15% more neurons in VGG-11, seen in Table VIII.

2) *Neuron coverage enhancement*: To evaluate the performance of FuzzGAN on enhancing neuron coverage during the testing phase, we set an upper limit of the size of test suite (100 for LeNet-1 and 1000 for VGG-11) and applied FuzzGAN to testing each DNN. We observed the maximum neuron coverage that FuzzGAN can achieve and compare the

TABLE VIII
NEURON OUTPUT ENHANCEMENT.

Net.	Output	No. of activation
LeNet-1	76.22%	108.90%
VGG-11	42.95%	146.15%

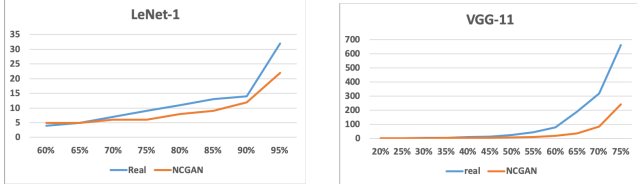


Fig. 4. The number of test cases required to achieve specific neuron coverage.

numbers of generated test cases or natural samples to reach the same neuron coverage.

As a result, with at most 100 test cases, FuzzGAN can achieve a neuron coverage of 100% (threshold $t = 0.6$) for testing a LeNet-1; with at most 1000 test cases, FuzzGAN can achieve a neuron coverage of 78.6% ($t = 0.4$). Compared with real data, it is proved that FuzzGAN needs less number of test cases to reach the same neuron coverage (shown in Fig. V-C2).

D. Improving DNN performance

In addition to uncovering adversarial vulnerabilities within tested DNN, adversarial examples generated by FuzzGAN can be adopted to enhance the robustness of this DNN by adversarial retraining [4]. We generated 2000 adversarial examples for each tested network. We augmented each training set of MNIST and LSUN by adding 1000 adversarial samples, retrained tested DNNs separately by 5 epochs.

The growth trend of the accuracy of retrained models on the original test sets in MNIST and LSUN are shown in Fig. 5. Additionally, we compared the performance of newly trained networks and previous networks with test sets containing 1000 adversarial examples and 2000 real samples. As a result, the retrained LeNet-1 and VGG-11 networks achieved 37.12% and 23.70% more average accuracy. Such results illustrate test cases generated by FuzzGAN can fix erroneous behaviors to some extent and are effective for improving the performance of the test DNN.

VI. RELATED WORK

In this section, we review the related work in the following two areas: adversarial attack and DNN test. The work on adversarial attack can be regarded as a predecessor of DNN test, since the test cases in test are adversarial examples as well, and DNN test in a sense is adversarial attack with additional requirements, such as maximizing test coverage.

A. Adversarial Attack

Szegedy et al. [24] firstly discovered that a slight perturbation added to the picture can deceive neural networks. This

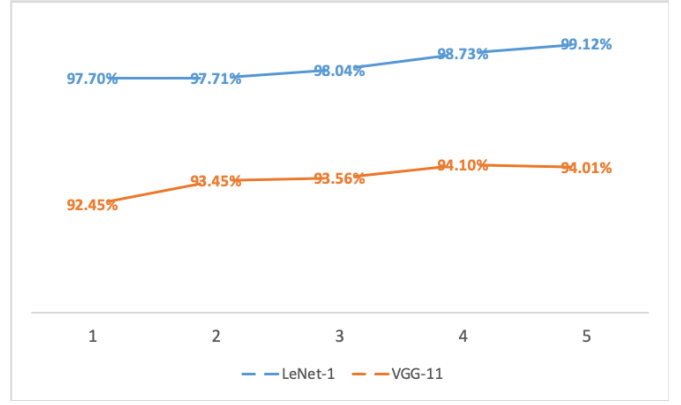


Fig. 5. The accuracy of retrained models.

process is called an adversarial attack, and a sample that is finely crafted by an adversary and can make model incorrectly classify is called an adversarial example.

Subsequently, many improved methods of generating adversarial example have been proposed, including FGSM (fast gradient sign method) [3], BIM (Basic Iterative Method) and ILCM (Least-Likely-Class Iterative Methods) [7], JSMA (Jacobian-based Saliency Map Attack) [15], one-pixel attack [22], C&W [1] and DeepFool [12] in white-box scenario.

Furthermore, Moosavi-Dezfooli et al. [13] generate universal adversarial perturbations that can be added to any image in ImageNet dataset and deceive different DNNs, including black-box ones. According to the cross-model transferability, Papernot et al. [16] provide another way to launch black-box adversarial attack by training a substitute model.

Unlike the above work generating adversarial examples by perturbing seed samples, advGAN [25] as a representative of GAN-based adversarial attacks generate adversarial examples against DNNs from random noise by generative adversarial networks.

B. DNN fuzzing

Adversarial examples generated by techniques mentioned above are similar to random test case (without considering any coverage criterion) for evaluating the robustness of DNN. Pei et al. [17] firstly introduced neuron coverage and proposed DeepXplore to systematically test DNNs. Subsequently, several recent work discussed the test coverage criteria of DNN: [10], [11], [23] further discussed the output of neurons in DNN and proposed delicately tailored test coverage criteria and corresponding test case generation algorithms. All these criteria are based on neuron output, and the basic idea of test case generation can be summarized as generating adversarial examples by adding perturbations to specific natural samples. Besides, experts from Google Brain proposed TensorFuzz [26] as a tool for DNN fuzzing that takes advantage of fast approximate nearest neighbor algorithms to measure test coverage. Xie et al. extended existing fuzzing technique with eight semantic-preserving metamorphic mutation types and implemented their fuzzing framework DeepHunter [26] for

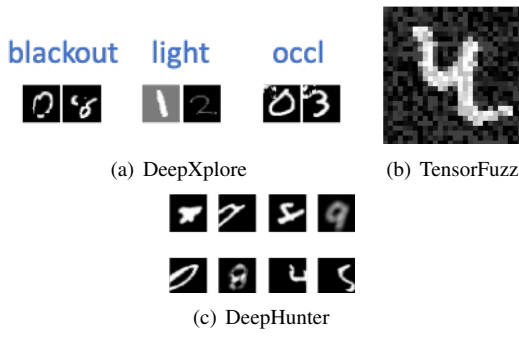


Fig. 6. Test cases generated by (a) DeepXplore, (b) TensorFuzz and (c) DeepHunter.

neuron coverage and other coverage criteria proposed in DeepGauge [10].

All of the work on DNN fuzzing mentioned above are based on mutation. The test cases are generated by pixel value transformation (including changing image brightness, blur, noise, etc.) and affine transformation (including image shearing, rotation, scaling, etc.). Mutation-based fuzzing two major weaknesses. Firstly, the test cases are generated by mutation so that there may exist significant flaws in images and the generated images are pretty easy to be distinguished by human, as shown in Fig. VI-B. Compared with such fuzzing technique, the test cases generated by FuzzGAN (Fig. IV) are more sharp, clear and realistic.

Secondly, since seed samples for fuzzer to mutate are settled, the diversity of generated images is supposed to be limited. Furthermore, a small number of transformations brought to a image may not lead the tested DNN to errors, which means changing an innocent real image to an adversarial example may requires dozens or even hundreds of transformations. As a result, mutation-based fuzzing may generate a mass of useless samples (non-adversarial) during testing. On the contrary, the rate of FuzzGAN generated data leading to errors is more than 53.5%, which shows a generation-based fuzzing will generates much less waste than a mutation-based one.

VII. CONCLUSION

We propose FuzzGAN that, under the guidance of a coverage criterion, generates valid, realistic and adversarial test cases from scratch to evaluate the robustness of DNNs against adversarial examples. To the best of our knowledge, FuzzGAN is the first generation-base fuzzing framework for testing DNNs. It synthesizes adversarial input samples that are indistinguishable from real ones by modeling the distribution of target dataset rather than mutation-based fuzzing that applies slight distortion to specified real samples as previous work. Additionally, for increasing the neuron coverage, the generator in FuzzGAN is designed in accordance with the concrete architecture of target DNN to generate inputs that maximize the output of specified neurons in tested DNN.

We have implemented FuzzGAN to test LeNet-1 on MNIST and VGG-11 on LSUN. The results show that the test cases generated by FuzzGAN can confuse human and can make

the test DNNs improperly classify at a probability more than 53.5% (75.5% for VGG-11). Moreover, it is demonstrated that the adversarial examples generated by FuzzGAN can not only be used as test cases to evaluate DNN robustness, but also be used for DNNs to defense adversarial attacks by adversarial retraining with augmented training set.

In the future, FuzzGAN can be extended to generate test cases of different formats like audio, video, malware, files, etc., and can be further applied to other coverage criteria.

REFERENCES

- [1] N. Carlini, and D. Wagner, "Towards evaluating the robustness of neural networks," in 2017 IEEE Symposium on Security and Privacy (S&P), pages 39-57, 2017.
- [2] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in Advances in neural information processing systems (NIPS), pages 2672-2680, 2014.
- [3] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," arXiv preprint arXiv:1412.6572, 2014.
- [4] K. Grosse, P. Manoharan, N. Papernot, M. Backes, and P. McDaniel, "On the (Statistical) Detection of Adversarial Examples," arXiv preprint arXiv:1702.06280, 2017.
- [5] J. Li, B. Zhao, and C. Zhang, "Fuzzing: a survey," Cybersecurity 1.1 (2018): 6, 2018.
- [6] C. Kaner, "Exploratory testing," Quality assurance institute worldwide annual software testing conference, 2006.
- [7] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," arXiv preprint arXiv:1607.02533, 2016.
- [8] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, 86.11: 2278-2324, 1998.
- [9] Y. LeCun, "The MNIST database of handwritten digits," <http://yann.lecun.com/exdb/mnist/>, 1998.
- [10] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li et al, "Deepgauge: Multi-granularity testing criteria for deep learning systems," in Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, pages 120-131. ACM, 2018.
- [11] L. Ma, F. Zhang, M. Xue, B. Li, Y. Liu, and J. Zhao, et al, "Combinatorial Testing for Deep Learning Systems," arXiv preprint arXiv:1806.07723, 2018.
- [12] S.M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2574-2582, 2016.
- [13] S. M. Moosa-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," arXiv preprint arXiv:1705.09554, 2017.
- [14] A. Odena and I. Goodfellow, "Tensorfuzz: Debugging neural networks with coverage-guided fuzzing," arXiv preprint arXiv:1807.10875, 2018.
- [15] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in 2016 IEEE European Symposium on Security and Privacy (EuroS&P), pages 372-387, 2016.
- [16] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, pages 506-519, 2017.
- [17] 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, pages 1-18, 2017.
- [18] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," arXiv preprint arXiv:1606.03498, 2016.
- [19] P. Santiago, A. Bonafonte, and J. Serra, "SEGAN: Speech enhancement generative adversarial network," arXiv preprint arXiv:1703.09452, 2017.
- [20] J. Schmidhuber, "Deep learning in neural networks: An overview," Neural networks 61:85-117, 2015.
- [21] K. Simonyan, and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv:1409.1556, 2014.

- [22] J. Su, D. V. Vargas, and S. Kouichi, "One pixel attack for fooling deep neural networks," *IEEE Transactions on Evolutionary Computation*, 2019.
- [23] Y. Sun, X. Huang, and D. Kroening, "Testing Deep Neural Networks," *arXiv preprint arXiv:1803.04792*, 2018.
- [24] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, and I. Goodfellow et al, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.
- [25] C. Xiao, B. Li, J. Y. Zhu, W. He, M. Liu, and D. Song, "Generating adversarial examples with adversarial networks," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence, AAAI Press (2018): 3905-3911*, 2018.
- [26] X. Xie, L. Ma, F. Juefei-Xu, H. Chen, M. Xue, B. Li, et al, "Coverage-Guided Fuzzing for Deep Neural Networks," *ArXiv e-prints*, September 2018.
- [27] F. Yu, Y. Zhang, S. Song, A. Seff, and J. Xiao, "Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop," *arXiv preprint arXiv:1506.03365*, 2015.
- [28] Y. Zhang, Z. Gan, and L. Carin, "Generating text via adversarial training," *NIPS workshop on Adversarial Training*, Vol. 21, 2016.
- [29] B. Zhao, B. Chang, Z. Jie, and L. Sigal, "Modular Generative Adversarial Networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 150-165, 2018.