2022 IEEE 24th Int Conf on High Performance Computing & Communications; 8th Int Conf on Data Science & Systems; 20th Int Conf on Smart City; 8th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application

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

Ge Han
School of Computer Science and Technology
Shandong University
Shandong, China
hangehg@126.com

Zheng Li
CISPA Helmholtz Center for
Information Security
Saarland, Germany
zheng.li@cispa.de

Peng Tang
School of Cyber Science and Technology
Shandong University
Shandong, China
tangpeng@sdu.edu.cn

Chengyu Hu
School of Cyber Science and Technology
Shandong University
Shandong, China
hcy@sdu.edu.cn

Shanqing Guo
School of Cyber Science and Technology
Shandong University
Shandong, China
guoshanqing@sdu.edu.cn

***Abstract*—Deep neural networks (DNNs) are increasingly deployed in various fields. Despite their spectacular advances, DNNs are known to suffer from adversarial vulnerabilities. The robustness of DNNs is then threatened by leading them to misclassifications with unexpected inputs (adversarial examples). The fuzzing technique frequently used for testing traditional software has recently been adopted to evaluate the robustness of DNNs. Current DNN fuzzing techniques focus on image classification DNNs and generate test cases by mutations, e.g., image transformations and adversarial perturbations. However, mutation-based test cases usually lack diversity and have distribution deflection from the original DNN input space, which impacts the evaluation of DNNs.**

In this paper, we propose a generation-based fuzzing framework FuzzGAN to detect adversarial flaws existing in DNNs. We integrate the testing purpose and the guidance of the neuron coverage into the original objectives of auxiliary classifier generative adversarial networks. Hence, FuzzGAN learns the representation of a DNN's input space and generates test cases without the limitation of any concrete seed input. We evaluate the performance of FuzzGAN on two DNN models that have classical network structures and are trained on public datasets. The experiment results demonstrate that FuzzGAN can generate realistic, diverse and valid test cases and achieve high neuron coverage. Moreover, these test cases can be used to improve the performance of the target DNN through adversarial retraining.

I. Introduction

Deep neural networks (DNNs) are a set of machine learning algorithms modelled loosely after the biological neural networks to progressively approach their tasks based on data representation learning [27]. The widespread adoption makes it crucial to discuss the security properties of DNNs, especially those adopted in safety-critical domains.Unfortunately, DNNs always suffer from adversarial attacks [31] which subtly modify inputs resulting in incorrect behaviours. Therefore, it is in great demand to test and improve the robustness of DNNs against adversarial attacks.

Recently, *fuzzing*, one of the most widely-deployed techniques for traditional software, has been utilized to evaluate DNN robustness [4], [6], [8], [22], [25], [33], [34], [38]. It tests DNNs by automatically generating misleading test cases guided by coverage criterion to find internal vulnerabilities. Furthermore, all current DNN fuzzing works are mutation-based fuzzing [13]. They apply various mutations to given seed inputs for generating test cases, such as perturbations [8], [22], [25], [33], image transformations [4], [6], [34] and image translation [38].

However, mutation-based fuzzing has two drawbacks in test case generation. Firstly, the diversity of test cases is limited since each test case is mutated from individual seed input, and seed inputs are settled. The lack of diversity may further affect the adequacy of detecting DNN vulnerabilities [1]. Secondly,converting an innocent image to adversarial for testing usually requires vast mutation and causes noticeable distribution deflection. Such deflection may damage the semantic information of test cases and confuse the testing – whether the misclassifications are caused by the distribution deflection in test cases or the defects of DNNs [13].

To mitigate the drawbacks of existing mutation-based DNN fuzzing techniques, we propose a new fuzzing framework FuzzGAN for evaluating DNN robustness. FuzzGAN is challenged to improve the quality of test cases with a novel generation method rather than seed input mutations. Secondly, a corresponding test oracle that determines the correctness of DNN behaviors is requisite for new test case generation method. Moreover, FuzzGAN is required to guarantee the adequacy of the testing, i.e. to maximize the coverage of tested DNNs effectively. In this paper, we take neuron coverage [25] as the coverage criterion to guide test case generation. The successful application of neuron coverage in FuzzGAN potentially implies a similar success in any other criterion.

To solve the challenges aforementioned, we design FuzzGAN as a generation-based fuzzing framework. Taking advantage of auxiliary classifier generative adversarial network (ACGAN) [23] technique, the test case generator in FuzzGAN

979-8-3503-1993-4/22/$31.00 ©2022 IEEE
DOI 10.1109/HPCC-DSS-SmartCity-DependSys57074.2022.00244

2022 IEEE 24th Int Conf on High Performance Computing & Communications; 8th Int Conf on Data Science & Systems; 20th Int Conf on Smart City; 8th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys) | 979-8-3503-1993-4/22/$31.00 ©2022 IEEE | DOI: 10.1109/HPCC-DSS-SMARTCITY-DEPENDSYS57074.2022.00244

learns the representation of real data and generates new data cases under specified classes. Due to the generative ability of generative adversarial networks [10], the generated cases are not limited to individual seeds and are located close to the real data distribution. The classes specified during generation are utilized as the test oracle to assess the correctness of DNN predictions on generated cases. Moreover, we take the tested DNN as an auxiliary party for training the test case generator. Besides fooling a discriminative network, the generator aims to lead the tested DNN to misclassifications and increase the output of the specified neuron. Then, generated cases are more likely to be adversarial for detecting DNN vulnerabilities and be able to enhance neuron coverage.

The main contributions of this work include:

- We propose a generation-based fuzzing framework FuzzGAN for evaluating the robustness of DNNs. To the best of our knowledge, FuzzGAN is the first generation-based DNN fuzzing framework so far.
- Taking advantage of generative adversarial networks, FuzzGAN achieves the following objectives: 1) generate realist images for specified class from random noise, 2) generate images leading the tested DNN to errors, 3) achieve high test coverage by a set of test cases (test suite).
- We implement FuzzGAN to separately test a LeNet-1 model trained on MNIST and a VGG-11 model on LSUN, and then evaluate it by comparing with related DNN fuzzers. The results present that the test cases generated by FuzzGAN are visually better than mutation-based test cases and have comparable performance on IS and FID metrics and on coverage enhancement. Additionally, it is proved that a pre-trained generation-based fuzzer is much more efficient than mutation-based fuzzers in generating adversarial test cases.

II. Preliminaries

A. Fuzz Testing for DNNs

Fuzz testing, often known as fuzzing, is a testing approach usually used in programming and software development. It automatically generates invalid, unexpected, or random data as inputs for a computer program trying to cause crashes, errors, memory leaks, and so on. The program is then detected for defects and vulnerabilities. Fuzz testing usually requires three primary components [2]: the **test coverage criterion** as metric for measuring the adequacy of testing and guide the generation of test cases, the **test oracle** for determining whether the output of target program is correct, and the **test case generation algorithm** describing the automatic process of generating test cases. Fuzzing techniques can be classified into **mutation-based fuzzing** and **generation-based fuzzing** according to whether the test cases are generated through the mutation of given inputs or generated by a specification or other previous knowledge [13].

The fuzzing technique has been used for evaluating the adversarial robustness of DNNs in recent work [4], [6], [8], [22], [25], [33], [34], [38]. The target of **DNN fuzzing** is to discover inner vulnerabilities that have the potential to be leveraged by adversarial attacks [31]. Targeting at classification DNNs, a DNN fuzzer tests a DNN by generating adversarial inputs as test cases and monitoring incorrect classifications.

B. Neuron Coverage

We take neuron coverage [25] as the coverage criterion of our fuzzing framework. The **neuron coverage** is defined as the ratio of the neurons activated by any test case in a test suite $T = \{x_0, x_1, ..., x_q\}$ (a set of test cases) to all the neurons $N = \{n_1, n_2, ..., n_p\}$ in the DNN under test. A neuron is considered to be activated if the value of its output is high enough (higher than a threshold t), such that its impact on neurons in subsequent layers and even the output layer is significant. Let $n_i(x_j)$ be the output value of a neuron n_i in DNN for a given test case x_j. The neuron coverage (NC) can be presented as follow:

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

Both positive and negative values exist among neuron outputs and weights. Hence, we regard that a larger absolute value of neuron output implies a more significant impact on the following layers, and a zero implies no impact. Additionally, we normalize the averaged output of all elements inside a multidimensional neuron as its output value. Hence, the neurons with various dimensions and value ranges can be compared with the same threshold.

C. Generative Adversarial Network Techniques

The auxiliary classifier generative adversarial network [23], or ACGAN for short, is an essential part of FuzzGAN. Two primary components within an ACGAN are a generator G and a discriminator D. The generator takes a latent code $z \sim p_z$ as input and attempts to produce a fake sample $x' = G(z)$, while the discriminator determines whether an observed sample x_in is real or fake. The competition between G and D enables mapping latent codes in a low-dimensional random distribution p_z to semantically meaningful samples in a high-dimensional space p_{data}. Especially, ACGAN is a type of GANs that involves the conditional generation of images with a corresponding class label $c \in C$ (available classes) in addition to the noise z: $x' = G(z|c)$. In addition to predicting whether a given image is real or fake, the discriminator in ACGAN predicts the class label of the given image. It allows FuzzGAN to generate test cases with labels as the test oracle.

III. Methodology

In this paper, we propose FuzzGAN to test the robustness of deep neural networks against adversarial examples. We investigate DNN testing in a white-box scenario where the tester has full access to the tested DNN, including the training algorithm, data, network structure, and internal weights. As a fuzzing framework, FuzzGAN automatically provides adversarial inputs as test cases for a DNN and monitors incorrect classification outputs. Generally, FuzzGAN employs

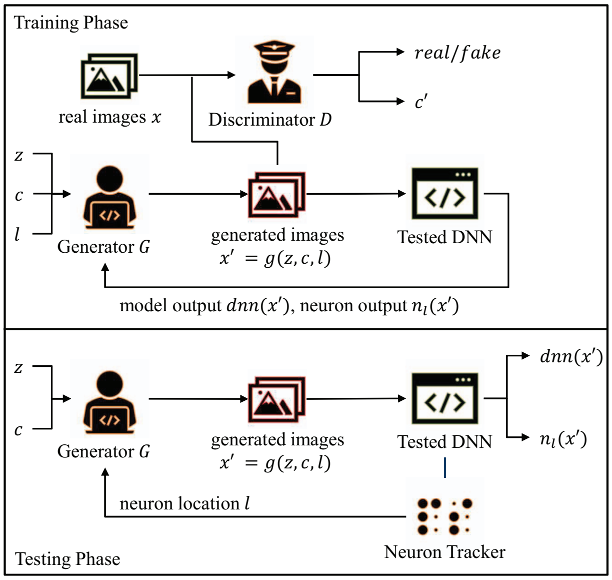


Fig. 1. An overview of the generation-based DNN fuzzing framework FuzzGAN.

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.
Function COVERAGE($neuron\_tracker$):
    $a\_num \leftarrow$ the number of activated neurons;
    $num \leftarrow$ the number of all neurons in tested DNN;
    return $a\_num/num$;
$test\_suite \leftarrow$ an empty set;
$neuron\_tracker \leftarrow$ set all neurons non-activated;
for COVERAGE($neuron\_tracker$) $< d$ do
    $c \leftarrow$ target class randomly selected from $C$;
    $z \leftarrow$ random noise;
    $l \leftarrow$ the location information of a non-activated
      neuron randomly selected from $neuron\_tracker$;
    $c \leftarrow$ target class randomly selected from $C$;
    $x' \leftarrow g\ (z, c, l)$;
    if $dnn\ (x') \neq c$ then
        $test\_suite$.add $(x')$;
        $neuron\_tracker$.update $(dnn, x')$;
```

the ACGAN technique to generate input cases under given class labels. The generation is under the guidance of neuron coverage. The labels given for generation are used as the test oracle to determine whether the predictions of a DNN on generated cases are correct. In this section, we start by introducing the general process of FuzzGAN, and then we describe the design of our test case generator.

A. *General Process of FuzzGAN*

Aiming to improve the diversity of generated test cases, we introduce generative adversarial network techniques to our fuzzing framework. The lifecycle of FuzzGAN contains two phases, i.e. training and testing phases (as shown in Figure 1).

In the training phase, a generative model G is deliberately trained against a discriminative model D. The generator is trained to learn the distribution of original inputs and synthesize samples addressing the challenges listed in section I. The details of G will be specifically described in the next subsection. The training phase can be regarded as a preparation for the subsequent testing phase, in which fuzz testing is conducted with the trained generative model as a test case generator.

In the testing phase (illustrated in algorithm 1), FuzzGAN iteratively generates input samples toward the tested DNN. We define a tracker $neuron\_tracker$ to record the activation state of all neurons in tested DNN. 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. The neuron n_l is randomly selected among non-activated neurons from $neuron\_tracker$. Then, G outputs a synthesized input sample $x' = g(z, c, l)$ to the tested DNN dnn.

Only the generated input samples that lead the tested DNN to erroneous classifications ($dnn(x') \neq c$) are effective test cases for fuzzing and will be appended into a test suite $test\_suit$. Once the test suite is enlarged by a new test case x', the neuron activation tracker $neuron\_tracker$ will be checked and updated. The loop of generating input samples terminates when the desired neuron coverage requirement d is achieved. In the end, the tester obtains a set of test cases $test\_suite$ through FuzzGAN, by which the test may achieve a relatively high neuron coverage.

B. *Test Case Generator*

Since FuzzGAN is proposed as a framework for generation-based fuzzing, its test case generator is expected to synthesize test cases by learning the global distribution of DNN input space rather than mutating from individual seeds. To this end, the generative adversarial network technique is employed to train it (displayed in Figure 1).

In the duration of the training procedure, FuzzGAN iteratively trains the test case generator G with a discriminative model D. The generator G takes random noise z, randomly selected class c and the location l of a randomly selected neuron n_l as input, and then synthesizes an input sample x' to the tested DNN dnn. The discriminative model D takes input x_{in} from either the real data set X or G generated samples, and then classifies x_{in} as c' and distinguishes whether the it is a real sample x ($x_{in} \in X$) or a generated one x' ($x_{in} = g(z, c, l)$). Both G and D are multi-layer perceptrons with convolutional layers. The training process of them is guided by the loss functions representing each model's objective. The models achieve their functionalities by decreasing the loss with the gradient descent method.

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

$$\begin{aligned} L_D &= L_{class} + L_{real} \\ L_{class} &= CrossEntropyLoss(p_c, c) \\ L_{real} &= E[\log P(real|x_{in} \in X)] \\ &\quad + E[\log P(fake|x_{in} \in g(z, c, l))] \end{aligned} \tag{2}$$

We use $p_c = \{p1, p2, ..., pk\}$ to denote a vector of probabilities classifying x_{in} as each class, where k is the number of classes. We use the cross-entropy loss $CrossEntropyLoss(\cdot)$ between p_c and c to represent D's loss on classifying L_{class}. It increases when the predicted probability diverges from the target class. L_{real} denotes the loss of determining whether an input is from the real data set, where $E(\cdot)$ calculates the value of expectation.

The G, firstly, aims to generate valid input cases under given classes, which requires confusing D's discrimination functionality and coordinating D's classifying. Thus, G is trained to maximize D's loss of discrimination L_{real} and to minimize L_{class}. Additionally, the generated data is supposed to address the rest objectives: leading the tested DNN to erroneous behaviours and maximising the neuron coverage. We take the tested DNN dnn as an auxiliary party and iteratively train G with the aid of feedback from it. The former DNN-related objective requires G to maximise the cross-entropy loss of dnn for not classifying the x' to the specified class c. At last, G is trained to maximise the output of the neuron specified by input l. l denotes a neuron's location information, including the layer l_{layer} it belongs to and the concrete index l_{index} inside this layer. When G is used for testing, the l will be assigned according to a traced neuron activation state.

In summary, the loss function of G is designed as a linear combination of three parts:

$$\begin{aligned} L_G &= L_{valid} + \lambda_1 \cdot L_{error} + \lambda_2 \cdot L_{neuron}, \\ L_{valid} &= L_{class} - L_{real}, \\ L_{error} &= -CrossEntropyLoss(dnn(x'), c), \\ L_{neuron} &= CrossEntropyLoss(layer(x', l_{layer}), l_{index}) \end{aligned} \tag{3}$$

$CrossEntropyLoss(\cdot)$ is used to measure the performance of the tested DNN on classification (L_{error}). It is also employed in L_{neuron}. Function $layer(x', l_{layer})$ returns the outputs of all neurons in the layer containing n_l. By minimizing L_{neuron}, the averaged absolute output value of the specified neuron n_l (neuron no.l_{index}) diverges from the others in the same layer (layer no. l_{layer}). Then, the normalized output value of n_l will be increased. 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.

IV. Implementation

As a proof-of-concept, we implement FuzzGAN for testing two DNNs. The first test DNN is a LeNet-1 [11] 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 [12]) with an accuracy of 98.34%. The input images to LeNet-1 are of size 28*28.

FuzzGAN has been further implemented to test a VGG-11 network [28] trained on dataset LSUN [35]. The images in LSUN are divided into 10 categories, and the VGG-11 under test was trained to classify images from four of them (towers, restaurants, dining rooms, and bridges). The tested VGG-11 takes RGB images in the size of 64*64 as input and achieves an accuracy of 93.4%. The VGG-11 contains 15 layers and 4763 neurons.

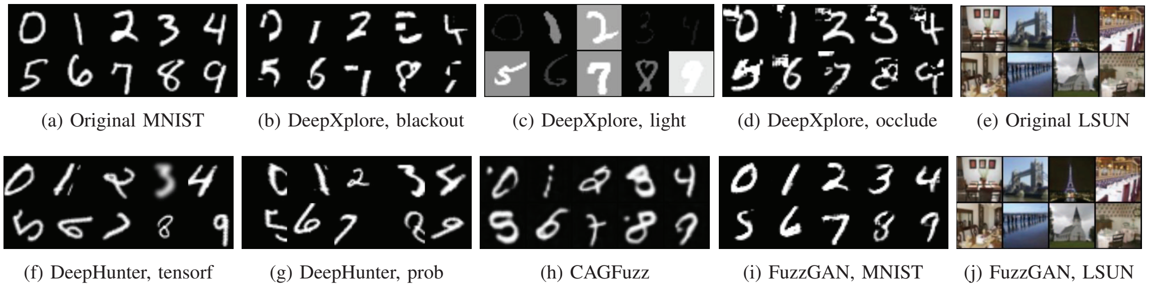
In the experiment, FuzzGAN is trained by traversing MNIST for 100 epochs (1200 epochs for LSUN) with a learning rate of 0.0002 (0.0004 for LSUN). Images from the dataset were fed in batches of 128, and one fed batch triggers one iteration of training steps. We performed the grid search to find the optimum loss weights λ_1 and λ_2 for each tested DNN. We empirically set λ_1 as 0.5 for testing the LeNet-1 on MNIST and 0.1 for the VGG-11 on LSUN. The value of λ_2 varies for each layer of a tested DNN.

V. Evaluation

We implement FuzzGAN to test two DNNs separately and analyse the testing results. Additionally, we take DeepXplore [25], DeepHunter [34] and CAGFuzz [38] among existing DNN testing techniques (reviewed in section VI) as our evaluation benchmarks. Since DeepXplore generates test cases in three different manners (blackout, light and occlude) and DeepHunter has two strategies for mutation selection (tensorfuzz and prob), we consider them as distinct benchmark fuzzers. Similarly to FuzzGAN, all the benchmark fuzzers employ the fuzzing technique, use neuron coverage as their coverage criterion and test a LeNet-1 trained on MNIST. Hence, we compare FuzzGAN with benchmarks on LeNet-1 and MNIST. The evaluation and comparison are made from three perspectives: image quality, misleading ability and neuron coverage enhancement. At the end of this section, we further discuss the improvement of tested DNN with adversarial retraining.

A. Image Quality

The primary motivation of FuzzGAN is to improve the image quality of generated test cases. As discussed before, the critical problems of test cases are distribution deflection and lack of diversity. Firstly, unrealistic test cases generated via inflated mutations may be filtered at a pre-processing stage of a DNN system. It is hard to determine whether an incorrect classification on such a test case is caused by DNN vulnerabilities or the deflection of test case distribution. Additionally, mutation-based test cases are limited to corresponding seed inputs, impacting the efficiency of covering neurons. In this subsection, we measure the image quality of test cases


(a) Original MNIST (b) DeepXplore, blackout (c) DeepXplore, light (d) DeepXplore, occlude (e) Original LSUN

(f) DeepHunter, tensorf (g) DeepHunter, prob (h) CAGFuzz (i) FuzzGAN, MNIST (j) FuzzGAN, LSUN

Fig. 2. Original images and test cases generated by different DNN fuzzers.

generated by FuzzGAN in a qualitative manner and with two quantitative metrics.

At first, **We discuss the reality of test cases in a visualized manner.** We carry out an evaluation with human subjects to demonstrate the indistinguishability of test cases against original data. For each of the DNNs trained on MNIST and LSUN, FuzzGAN learns the representation of its original inputs and generates test cases with similar representation. We randomly mix 20 generated images with 20 real images for each dataset and construct a test set for our subjects. We invite 30 subjects to distinguish which images in the test set are synthetic without telling them the proportion of synthetic images. For hand-written-digit images, our subjects achieve an average accuracy of 49.17%, which seems to be randomly guessing. However, since images in LSUN display natural scenes with complex structures, our small-sized (64*64) training data are not enough for the generator in FuzzGAN to perfectly learn the representation of each scene. The performance of FuzzGAN on imitating LSUN is slightly weaker than imitating MNIST: the subjects distinguish the generated images from real ones with an accuracy of around 57.08%.

Furthermore, we compare the test cases generated by FuzzGAN and benchmark DNN fuzzers with real images sampled from MNIST and LSUN (shown in Figure 2). Subfigures (a) and (e) display the real images, and subfigures (i) and (j) are test cases generated by FuzzGAN. It can be intuitively found that test cases generated by FuzzGAN (subfigures (i) and (j)) look strongly like real images (subfigures (a) and (e)). Compared with the other test cases generated by DeepXplore, DeepHunter and CAGFuzz, we can conclude that the ones generated by FuzzGAN are more sharp, clear and realistic.

Additionally, we measure the image quality of generated test cases with two quantitative metrics. Inception Score (IS) [26] is a metric for evaluating the generating capability of a generative model. It focuses on two desirable traits of generating images: each containing one clear object and having high diversity. It leads to a large IS if both of these qualities are satisfied. We calculate IS and compare the dataset MNIST with test cases generated by FuzzGAN and benchmark fuzzers. As shown in Table I, the images generated by FuzzGAN have similar IS as real images, while most of the test cases generated by other fuzzers are of lower IS. It proves that FuzzGAN performs well on clarity and diversity.

TABLE I
COMPARISON OF TEST CASES GENERATED FROM THE ORIGINAL MNIST TEST SET AND DIFFERENT FUZZERS.

Test Cases	IS	FID	Adversarial Rate
MNIST	2.527	0	1.70%
DeepXplore_blackout	**2.505**	**32.651**	9.60%
DeepXplore_light	2.23	100.072	2.70%
DeepXplore_occlude	2.227	48.057	4.40%
DeepHunter_tensorf	1.84	77.692	8.10%
DeepHunter_prob	2.227	65.589	8.07%
CAGFuzz	1.984	63.895	3.60%
FuzzGAN	2.494	63.136	**53.50%**

Fréchet Inception Distance (FID) [9] is another popularly metric evaluating generated data. With a pre-trained Inception network extracting image features, FID measures the distance between the generated and real images at a feature level. Rather than IS focusing on clarity and diversity, FID captures the similarity of images. A low FID corresponds to a similar distribution of generated images to real images. As shown in Table I, the test cases generated by FuzzGAN are closer to the original dataset than the others, except for DeepXplore techniques with blackout and occlude perturbations.

It is worth noting that one DeepXplore achieves the best IS and FID among DNN fuzzers. It generates a test case by occluding a given image with a rectangle of noise (blackout). Although it has better performance on quantitative metrics, the images it generated (as shown in Figure 2 (b)) have a quite significant mark of occlusion and are very easy to recognize. This situation suggests that current quantitative metrics can measure the image quality to some extent but still are not enough to represent the visual difference between images.

B. Misleading Ability

The purpose of evaluating the adversarial robustness of DNNs is to discover potential vulnerabilities. DNN fuzzing techniques expose vulnerabilities by leading DNNs to misclassifications with elaborate inputs. The kernel of a DNN fuzzer is how to generate such adversarial inputs as test cases. A high probability of generating adversarial input samples always implies a high test efficiency. We calculate the percentage of adversarial samples in all samples generated by each DNN fuzzer. It shows that FuzzGAN has led the tested DNNs to

TABLE II
NEURON OUTPUT ENHANCEMENT.

Target DNN	Output Improvement	Activation Improvement
LeNet-1	76.22%	108.90%
VGG-11	42.95%	146.15%

misclassifications at a probability of more than 53.5% for LeNet-1 (75.5% for VGG-11), which is significantly more efficient than mutation-based fuzzers (shown in Table I). It is because the extent of each mutation step applied to the seed is commonly minute. Mutations are iteratively accumulated until the generated sample can lead the tested DNN to a misclassification. Hence, numerous intermediate samples will be generated during mutation-based fuzzing. Unlikely, FuzzGAN generates adversarial input cases directly by a pre-trained generator. Within the same time, FuzzGAN can generate much more test cases, which allows FuzzGAN to reach the desired coverage and terminal the test earlier than mutation-based fuzzers.

C. Neuron Coverage Enhancement

As introduced in subsection III-B, the generator is designed to synthesize input samples maximizing the output of the neuron specified by its input l. In the testing phase, neurons are specified according to a tracker of neuron coverage.

Firstly, we show that a FuzzGAN-generated data sample can enhance the output of the specified neuron or even activate it. We randomly select a set of neurons from each tested DNN (12 from LeNet-1 and 10 from VGG-11). We separately select 100 images from MNIST and LSUN as a control group and use FuzzGAN to generate 100 samples for each selected neuron as the experimental group. When feeding an image into a DNN, we record the output (normalized absolute values) and activation state for its corresponding neuron. We set the threshold of activation to 0.6. Statistical analysis shows that the outputs of selected neurons in LeNet-1 with FuzzGAN-generated samples are larger than that with MNIST images by approximately 76.22%. For the VGG-11, FuzzGAN effectively increases the output of selected neurons by around 42.95%. Furthermore, inputs generated by FuzzGAN averagely activate 108.90% more neurons in LeNet-1 than MNIST and activate 146.15% in VGG-11, seen in Table II.

Secondly, we implement FuzzGAN and observe the neuron coverage it achieves (shown in Figure 3 and Figure 4). We set an upper limit on the test suite size (30 for LeNet-1 and 700 for VGG-11) and apply FuzzGAN to test DNNs. As a result, FuzzGAN achieved a neuron coverage of 98.1% (threshold t = 0.6) for testing the LeNet-1 and a neuron coverage of 75% (t = 0.4) for testing the VGG-11. Compared with images from original datasets, FuzzGAN requires less number of test cases to reach the same neuron coverage. The performance of FuzzGAN on LeNet for MNIST is also compared with benchmark fuzzers (seen in Figure 3). According to our reproduced results, the advantage of FuzzGAN is not significant enough but is competitive with existing mutation-based fuzzers.

D. Improving DNN performance

In this subsection, we attempt to eliminate DNN vulnerabilities with the test cases that discover them. Test case generation of DNN fuzzing always has a requirement on test coverage. Taking neuron coverage as an example, we intuitively assume that the vulnerability uncovered by a test case can be attributed to the neurons it activates. Therefore, DNN fuzzing not only exposes adversarial vulnerabilities but also locates them.

Taking advantage of the adversarial retraining technique [7], we retrain the tested DNNs. We add 2000 adversarial examples into the original training set for each DNN. The DNNs are then retrained on augmented training sets for 5 epochs. Figure 5 shows the growth trends of the accuracy of retrained models on their test sets. We compared the performance of newly trained networks and previous networks with test sets containing 1000 test cases and 2000 real samples. As a result, the retrained LeNet-1 and VGG-11 networks achieved 37.12% and 23.70% higher accuracy on mixed test sets. It illustrates that test cases generated by FuzzGAN are effective for improving test DNN by fixing erroneous behaviours to some extent.

VI. RELATED WORK

Besides fuzzing, other testing techniques such as concolic testing [29], [30] and combinatorial testing [19] have also been used in testing DNNs. Table III chronologically compare FuzzGAN with existing DNN testing techniques from three perspectives: The column *Neuron Coverage* presents whether neuron coverage is employed to measure testing adequacy; the column *Generation Type* shows the basic method used for generating test cases; the column *Test Oracle* lists how the correctness of tested DNNs is determined.

Firstly, FuzzGAN uses neuron coverage to measure testing adequacy as the most relevant work. According to the characteristics of DNNs, existing test coverage criteria for traditional software are insufficient for DNNs. Many new criteria have been proposed specifically for testing DNNs. Pei et al. [25] introduced neuron coverage for measuring the adequacy of DNN testing. Then, several recent works further discussed the output of neurons and proposed delicately tailored test coverage criteria, such as neuron bound coverage [20], sign-sign coverage [29] and t-way combination sparse coverage [19]. Among all criteria, neuron coverage is the most fundamental and popularly used in DNN testing techniques.

Secondly, FuzzGAN proposes a novel test case generation method different from current mutation-based testing techniques. For example, DeepXplore [25] and DeepBillboard [39] generate test cases with constrained perturbations. DeepTest [32], DeepHunter [34], DeepSmartFuzzer [4] and Sensei [6] utilize image transformations. DeepGauge [20], DeepGini [5] and RobOT [33] take advantage of existing adversarial attacks (e.g. Jacobian-based Saliency Map Attack (JSMA) [24] and Carlini and Wagner attack (C&W) [3]). DeepConcolic [30] and DeepCover [29] formalize coverage criteria and adopt symbolic analysis to mutate seed inputs. TensorFuzz [22], DeepCT [19], DLFuzz [8] and Test4Deep [36] propose optimization algorithms to generated

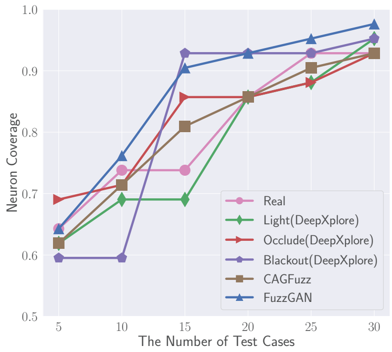


Fig. 3. Comparison of test cases obtained from different fuzzers and MNIST dataset in increasing the neuron coverage of target LeNet-1.

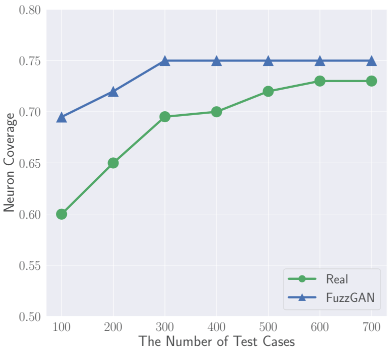


Fig. 4. Comparison of test cases obtained from FuzzGAN and LSUN dataset in increasing the neuron coverage of target VGG-11.

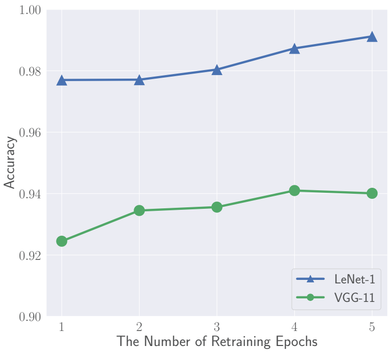


Fig. 5. Improvement in accuracy by retraining DNN models with test cases.

perturbations for test case generation. DeepMutation [21] mutates a target model rather than seed inputs.

To mitigate the influence of the image quality of test cases on DNN performance, DeepRoad [37] and CAGFuzz [38] adopt GANs to generate realistic test cases. They are similar to FuzzGAN, while the GANs they use are trained for image-to-image translation. According to DeepRoad and CAGFuzz, the test cases generated with GANs are more realistic than the cases generated by other mutations. However, image translation is still a kind of mutation in which a new image is generated from a given seed. Therefore, they still have a problem with the diversity of test cases. It's worth noting that DeepRoad does not consider coverage criterion, and CAGFuzz filters test cases based on the neuron coverage after generation.

At last, FuzzGAN provides a new test oracle. In existing testing techniques, test cases are mutated from seed inputs, so the predictions on test cases are related to the label of corresponding seed inputs. For example,DeepGauge [20] evaluates the predictions of a tested DNN on test cases by the labels of corresponding seed inputs. DeepTest [32] and DeepHunter [34] utilize transformation-specific metamorphic relations between test cases and seed inputs, and DeepGini [5] analyzes misclassification likelihood. Additionally, DeepXplore [25] requires multiple DNNs with the same functionality as a cross-reference oracle. Unlike existing techniques, FuzzGAN employs ACGAN to generate data cases under given class labels which are used as test oracle.

Besides evaluating the robustness of DNNs, there exists a wide range of other attacks, defenses and applications in machine learning domain [14]–[18]

VII. Conclusion

To mitigate the impact of the limited diversity and distribution deflection of mutation-based test cases, we propose a generation-based fuzzing framework FuzzGAN. It generates realistic, diverse and misleading test cases with the aid of ACGAN technique rather than applying mutations to given seeds as a mutation-based fuzzing. We guide the test case generation by neuron coverage and provide a valid test oracle for FuzzGAN.

To our knowledge, FuzzGAN is the first generation-based fuzzing framework for testing DNNs. We have implemented it to test a LeNet-1 on MNIST and a VGG-11 on LSUN. The results show that the test cases generated by FuzzGAN are realistic enough to confuse humans and have comparable IS and FID scores with those generated by other DNN fuzzers. Our test cases can enhance neuron coverage by improving the output of specified neurons and lead the tested DNNs to misclassifications at a much higher probability. Moreover, we show that the test cases generated by FuzzGAN are helpful in eliminating vulnerabilities by retraining tested DNNs.

References

[1] M. M. Almasi, H. Hemmati, G. Fraser, A. Arcuri, and J. Benefelds, "An Industrial Evaluation of Unit Test Generation: Finding Real Faults in a Financial Application," in *International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*. IEEE, 2017, pp. 263–272.

[2] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Transactions on Software Engineering*, 2014.

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

[4] S. Demir, H. F. Eniser, and A. Sen, "DeepSmartFuzzer: Reward Guided Test Generation For Deep Learning," CoRR abs/1911.10621, 2019.

[5] Y. Feng, Q. Shi, X. Gao, J. Wan, and C. Fang, "DeepGini: prioritizing massive tests to enhance the robustness of deep neural networks," in *International Symposium on Software Testing and Analysis (SIGSOFT)*. ACM, 2020, p. 177–188.

[6] X. Gao, R. K. Saha, M. R. Prasad, and A. Roychoudhury, "Fuzz Testing based Data Augmentation to Improve Robustness of Deep Neural Networks," in *International Conference on Software Engineering (ICSE)*. IEEE, 2020, pp. 1147–1158.

[7] K. Grosse, P. Manoharan, N. Papernot, M. Backes, and P. McDaniel, "On the (statistical) detection of adversarial examples," CoRR abs/1702.06280, 2017.

[8] J. Guo, Y. Jiang, Y. Zhao, Q. Chen, and J. Sun, "Dlfuzz: Differential fuzzing testing of deep learning systems," in *ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 2018, pp. 739–743.

TABLE III
COMPARISON OF ROBUSTNESS TESTING TECHNIQUES FOR DNNS.

Testing Technique	Neuron Coverage	Generation Type	Test Oracle
DeepXplore [25]	√	constrained perturbation	cross-referencing
DeepTest [32]	√	image transformation	metamorphic relation
DeepGauge [20]	√	adversarial perturbation	self-referencing
DeepMutation [21]	×	model mutation	self-referencing
DeepRoad [37]	×	image translation	metamorphic relation
DeepBillboard [39]	×	constrained perturbation	self-referencing
DeepConcolic [30]	√	symbolic mutation	self-referencing
DeepCover [29]	√	symbolic mutation	self-referencing
TensorFuzz [22]	√	optimized perturbation	numerical error
DeepCT [19]	×	optimized perturbation	self-referencing
DLFuzz [8]	√	optimized perturbation	self-referencing
DeepHunter [34]	√	image transformation	metamorphic relation
DeepSmartFuzzer [4]	√	image transformation	self-referencing
Sensei [6]	√	image transformation	self-referencing
DeepGini [5]	×	adversarial perturbation	error likelihood
RobOT [33]	×	adversarial perturbation	self-referencing
CAGFuzz [38]	√	image translation	self-referencing
Test4Deep [36]	√	optimized perturbation	self-referencing
FuzzGAN	√	**image generation**	**specified in generation**

[9] M. Heusel, H. R. andThomas Unterthiner, B. Nessler, G. Klambauer, and S. Hochreiter, "GANs Trained by a Two Time-Scale Update Rule Converge to a Nash Equilibrium," CoRR abs/1706.08500, 2017.

[10] A. Jabbar, X. Li, and B. Omar, "A Survey on Generative Adversarial Networks: Variants, Applications, and Training," in *Computing Surveys*. ACM, 2021, p. 1–49.

[11] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, pp. 2278–2324, 1998.

[12] Y. LeCun and C. C. C. J. Burges, "The MNIST database of handwritten digits," http://yann.lecun.com/exdb/mnist/.

[13] J. Li, B. Zhao, and C. Zhang, "Fuzzing: a survey," *Cybersecurity*, 2018.

[14] Z. Li, G. Han, S. Guo, and C. Hu, "Deepkeystego: Protecting communication by key-dependent steganography with deep networks," in *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, 2019, pp. 1937–1944.

[15] Z. Li, C. Hu, Y. Zhang, and S. Guo, "How to Prove Your Model Belongs to You: A Blind-Watermark based Framework to Protect Intellectual Property of DNN," in *Annual Computer Security Applications Conference (ACSAC)*. ACM, 2019, pp. 126–137.

[16] Z. Li, Y. Liu, X. He, N. Yu, M. Backes, and Y. Zhang, "Auditing Membership Leakages of Multi-Exit Networks," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2022.

[17] Z. Li, N. Yu, A. Salem, M. Backes, M. Fritz, and Y. Zhang, "UnGANable: Defending Against GAN-based Face Manipulation," CoRR abs/2210.00957, 2022.

[18] Z. Li and Y. Zhang, "Membership Leakage in Label-Only Exposures," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2021, pp. 880–895.

[19] L. Ma, F. Juefei-Xu, M. Xue, B. Li, L. Li, Y. Liu, and J. Zhao, "Deepct: Tomographic combinatorial testing for deep learning systems," in *International Conference on Software Analysis, Evolution,and Reengineering (SANER)*. IEEE, 2019, pp. 614–618.

[20] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu, Z. Jianjun, and W. Yadong, "Deepgauge: Multi-granularity testing criteria for deep learning systems," in *International Conference on Automated Software Engineering (ASE)*. IEEE, 2018, pp. 120–131.

[21] L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. Juefei-Xu, C. Xie, L. Li, Y. Liu, J. Zhao, and Y. Wang, "DeepMutation: Mutation Testing of Deep Learning Systems," in *IEEE International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2018, pp. 100–111.

[22] A. Odena and I. Goodfellow, "Tensorfuzz: Debugging neural networks with coverage-guided fuzzing," CoRR abs/1807.10875, 2018.

[23] A. Odena, C. Olah, and J. Shlens, "Conditional Image Synthesis with Auxiliary Classifier GANs," in *International Conference on Machine Learning (ICML)*. PMLR, 2017, pp. 2642–2651.

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

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

[26] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *Annual Conference on Neural Information Processing Systems (NeurIPS)*. MIT Press, 2016, pp. 2234–2242.

[27] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, 2015.

[28] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," CoRR abs/1409.1556, 2014.

[29] Y. Sun, X. Huang, and D. Kroening, "Testing deep neural networks," CoRR abs/1803.04792, 2018.

[30] Y. Sun, M. Wu, W. Ruan, X. Huang, M. Kwiatkowska, and D. Kroening, "Concolic testing for deep neural networks ," in *International Conference on Automated Software Engineering (ASE)*. IEEE, 2018, pp. 109–119.

[31] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," CoRR abs/1312.6199, 2013.

[32] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *International Conference on Software Engineering (ICSE)*. IEEE, 2018, pp. 303–314.

[33] J. Wang, J. Chen, Y. Sun, X. Ma, D. Wang, J. Sun, and P. Cheng, "RobOT: Robustness-Oriented Testing for Deep Learning Systems," in *International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1558–1225.

[34] X. Xie, L. Ma, F. Juefei-Xu, H. Chen, M. Xue, B. Li, Y. Liu, J. Zhao, J. Yin, and S. See, "Coverage-guided fuzzing for deep neural networks," CoRR abs/1809.01266, 2018.

[35] F. Yu, A. Seff, Y. Zhang, S. Song, T. Funkhouser, and J. Xiao, "LSUN: Construction of a large-scale image dataset using deep learning with humans in the loop," CoRR abs/1506.03365, 2015.

[36] J. Yu, S. Duan, and X. Ye, "A White-Box Testing for Deep Neural Networks Based on Neuron Coverage," *Transactions on Neural Networks and Learning Systems (TNNLS)*, 2022.

[37] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems," in *International Conference on Automated Software Engineering (ASE)*. IEEE, 2018, pp. 132–142.

[38] P. Zhang, B. Ren, H. Dong, and Q. Dai, "CAGFuzz:Coverage-Guided Adversarial Generative Fuzzing Testing for Image-based Deep Learning Systems," *Transactions on Software Engineering (TSE)*, 2021.

[39] H. Zhou, W. Li, Y. Zhu, Y. Zhang, B. Yu, L. Zhang, and C. Liu, "Deepbillboard: Systematic physical-world testing of autonomous driving systems," CoRR abs/1812.10812, 2018.