CAGFuzz: Coverage-Guided Adversarial Generative Fuzzing Testing of Deep Learning Systems

Pengcheng Zhang, *Member, IEEE*, Qiyin Dai, Patrizio Pelliccione

Abstract—Deep Learning systems (DL) based on Deep Neural Networks (DNNs) are increasingly being used in various aspects of our life, including unmanned vehicles, speech processing, intelligent robotics and etc. Due to the limited dataset and the dependence on manually labeled data, DNNs always fail to detect erroneous behaviors. This may lead to serious problems. Several approaches have been proposed to enhance adversarial examples for testing DL systems. However, they have the following two limitations. First, most of them do not consider the influence of small perturbations on adversarial examples. Some approaches take into account the perturbations, however, they design and generate adversarial examples based on special DNN models. This might hamper the reusability on the examples in other DNN models, thus reducing their generalizability. Second, they only use shallow feature constraints (e.g. pixel-level constraints) to judge the difference between the generated adversarial example and the original example. The deep feature constraints, which contain high-level semantic information - such as image object category and scene semantics, are completely neglected. To address these two problems, we propose *CAGFuzz*, a Coverage-guided Adversarial Generative Fuzzing testing approach for Deep Learning Systems, which generates adversarial examples for DNN models to discover their potential defects. First, we train an Adversarial Case Generator (*AEG*) based on general data sets. *AEG* only considers the data characteristics, and avoids low generalization ability. Second, we extract the deep features of the original and adversarial examples, and constrain the adversarial examples by cosine similarity to ensure that the semantic information of adversarial examples remains unchanged. Finally, we use the adversarial examples to retrain the model. Based on three standard data sets, we design a set of dedicated experiments to evaluate *CAGFuzz*. The experimental results show that *CAGFuzz* can improve the neuron coverage rate, detect hidden errors, and also improve the accuracy of the target DNN.

Index Terms—deep neural network; fuzz testing; adversarial example; coverage criteria.

1 Introduction

Nowadays, we have already stepped into the era of artificial intelligence from the digital era. Apps with AI systems can be seen everywhere in our daily life, such as Amazon Alexa [1], DeepMind's Atari [2], and AI-phaGo [3]. With the development of edge computing, 5G technology and etc., AI technologies become more and more mature. In many applications, we can see the shape of deep neural networks (DNNs), such as automatic driving [4], intelligent robotics [5], smart city applications [6] and AI-enabled Enterprise Information Systems [7]. In this paper, we term this kind of applications as DL (deep learning) systems.

In particular, many different kinds of DNNs are embedded in security and safety-critical applications, such as automatic driving [4] and intelligent robotics [5]. This brings new challenges since predictability, correctness, and safety are especially crucial for this kind of DL systems. These safety-critical applications deploying DNNs without comprehensive testing could have serious problems. For example, in automatic driving systems, if the deployed DNNs have not recognized the obstacles ahead timely and correctly, it may lead to serious consequences such as vehicle damage and even human death [8].

Generally speaking, the development process of DL systems is essentially different from the traditional software development process. As shown in Fig. 1, for traditional software development practices, developers directly specify the logic of the systems. On the contrary, DL systems automatically learn their models and corresponding parameters from data. Consequently, the testing process of DL systems is also different from traditional software systems. For traditional software systems, code or control-flow coverage is utilized to guide the testing process [9]. However, the logic of the DL systems is not encoded by control flow, and it cannot be solved by the normal encoding way. Their decisions are always made by training data for many times, and the performance is more dependent on data rather than human intervention. For DL systems, neural coverage can be used to guide the testing process [10]. When faults are found, it is also very difficult to locate the exact position in the original DL systems. Consequently, most traditional software testing methodologies are not suitable for testing DL systems. As highlighted in [10], [11], research on developing new testing techniques for DL systems is urgently needed.

The standard way to test DL systems is to collect and manually mark as much actual test data as possible [12],

- P. Zhang and Q. Dai are with the College of Computer and Information, Hohai University, Nanjing, P.R.China
  E-mail: pchzhang@hhu.edu.cn
- P. Pelliccione is with the University of L'Aquila, Italy and Chalmers | University of Gothenburg, Sweden.
  E-mail: patrizio.pelliccione@univaq.it

Manuscript received XXXX XXXX; revised XXXX, XXXX.

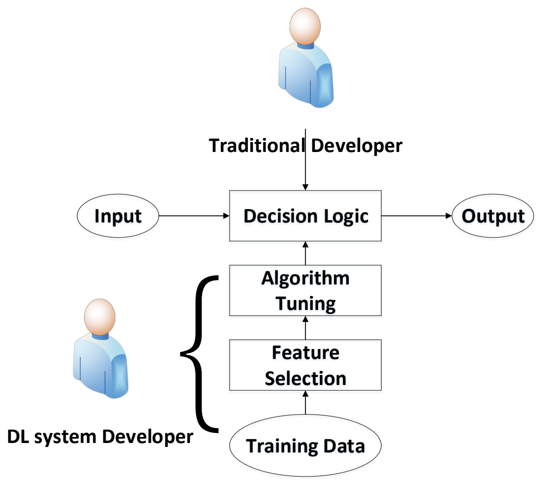


Fig. 1. Comparison between traditional and DL system development

[13]. Obviously, it is unthinkable to exhaustively test every feasible input of the DL systems. Recently, an increasing number of researchers have contributed to test DL systems with a variety of approaches [10], [11], [14], [15], [16]. The main idea of these approaches is to enhance input examples of test data set by different techniques. Some approaches, e.g. *DeepXplore* [10], use multiple DNNs to discover and generate adversarial examples that lie between the decision boundaries of these DNNs. Some approaches, e.g. *DeepHunter* [11], use metamorphic mutation strategy to generate new test examples. Other approaches, e.g. *DeepGauge* [16], propose new coverage criteria for deep neural networks. These coverage criteria can be used as guidance for generating test examples. While state-of-the-art approaches make some progresses on testing DL systems, they still suffer the following two main problems:

1) *DNN-dependent generation of adversarial examples.* Most approaches [11], [17] do not consider the influence of small perturbations on deep neural networks when the test examples are generated. Some approaches [10], [18] consider small perturbations based on special DNN models. The test examples that they have generated are only designed for one special DNN, and it may be difficult to generalize them to other DNNs. Recent research on adversarial DL systems [19], [20] shows that by adding the small perturbations to existing images and elaborating synthetic images can fool state-of-the-art DL systems. Therefore, to improve the generalization ability, it is significantly important to add small perturbations only based on data.
2) *Shallow feature constraints.* State-of-the-art adversarial example generation approaches use shallow feature constraints, such as pixel-level constraints, to judge the difference between the adversarial example and the original example. The deep feature constraints containing high-level semantic information, such as image object category and scene semantics, are completely neglected. For example, in their study, Xie et al. [11] use L_0 and L_∞ to limit the pixel-level changes of the adversarial example. However, such shallow feature constraints can only represent the visual consistency between the adversarial example and the original example, and cannot guarantee the high-level semantic information consistency between the adversarial example and the original example. Furthermore, this may lead to bad performance when testing the network with deep layers.

To address the problems aforementioned, we propose a new testing approach for DL systems, called *CAGFuzz* (Coverage-guided Adversarial Generative Fuzzing)[1]. The goal of the *CAGFuzz* is to maximize the neuron coverage and generate adversarial test examples as much as possible with small perturbations for the target DNNs. The goal of the *CAGFuzz* is to maximize the neuron coverage and generate adversarial test examples with minimal perturbations for the target DNNs. Meanwhile, the generated examples have strong generalization ability and can be used to test different DNN models. *CAGFuzz* iteratively selects the test examples in the processing pool and generates the adversarial examples through the pre-trained adversarial example generator (see Section 3 for details) to guide DL systems to expose incorrect behaviors. During the process of generating adversarial examples, *CAGFuzz* keeps valid adversarial examples, which can provide a certain improvement in neuron coverage for subsequent fuzzy processing, and limit the small perturbations invisible to human eyes, ensuring the same meaningfulness between the original example and the adversarial example. The contributions of this paper include the following three aspects:

- *We design an adversarial example generator, AEG, which can generate adversarial examples with small perturbations based on general data sets.* The goal of *CycleGAN* [21] is to transform *image A* to *image B* with different styles. Based on *CycleGAN*, our goal is to transform *image B* back to *image A*, and get *image A′* similar to the original *image A*. Consequently, we combine two generators with opposite functions of *CycleGAN* as our adversarial example generator. The adversarial examples generated by *AEG* can add small perturbations invisible to human eyes to the original examples. *AEG* is trained based on general data sets and does not rely on any specific DNN model, which has higher generalization ability than state-of-the-art approaches. Furthermore, because of the inherent constraint logic of *CycleGAN*, the trained *AEG* not only has high efficiency in generating adversarial examples but also can effectively improve the robustness of DL systems.
- *We extract the deep features of the original example and the adversarial example, and make them as similar as possible by similarity measurement.* We use VGG-19 network [22] to extract the deep semantic information of the original example and the adversarial example, and use the method of cosine similarity measurement to ensure that the deep semantic information of the adversarial example is consistent with the original

1. https://github.com/QXL4515/CAGFuzz

example as much as possible. At the same time, the deep feature constraint can make the adversarial examples generated by *CAGFuzz* get better results compared with other approaches when testing the network with deep layers.

- *We design a series of experiments to evaluate the CAGFuzz approach based on several public data sets.* The experiments validate that *CAGFuzz* can effectively improve the neuron coverage of the target DNN model. Meanwhile, it is proved that the adversarial examples generated by *CAGFuzz* can find hidden defects in the target DNN model. Furthermore, the accuracy and the robustness of the DNN models retrained by *AEG* have been significantly improved. For example, the accuracy of the VGG-16 [22] model in the experiments has been improved from the original 86.72% to 97.25%, with an improvement of 12.14%.

The rest of the paper is organized as follows. Section 2 provides some basic concepts including *CycleGAN*, Coverage-guided Grey-box Fuzzing (CGF). The coverage-guided adversarial generative fuzzy testing framework is provided in Section 3. In Section 4, we use three popular datasets (MNIST [23], Cifar-10 [24], and ImageNet [25]) to validate our approach. Existing work and their limitations are discussed in Section 5. Finally, Section 6 concludes the paper and looks into future work.

2 Preliminaries

The principles of coverage-guided grey-box fuzzing, *CycleGAN*, and VGG-19 are introduced in Section 2.1, Section 2.2, and Section 2.3, respectively. Section 2.4 introduces the basic concept and calculation formula for neuron coverage.

2.1 Coverage-guided Grey-box Fuzzing

Due to the scalability and effectiveness in generating useful defect detection tests, fuzzing has been widely used in academia and industry. Based on the perception of the target program structure, the fuzzy controller can be divided into black-box, white-box, and grey-box. One of the most successful techniques is Coverage-guided Grey-box Fuzzing (CGF), which balances effectiveness and efficiency by using code coverage as feedback. Many state-of-the-art CGF approaches, such as AFL [26], libFuzzer [27], and VUzzer [28], have been widely used and proved to be effective. Smart Greybox Fuzzing (SGF) [29] has made some improvements on CGF. Specifically, it leverages a high-level structural representation of the original example to generate new examples. The state-of-the-art CGF approaches mainly consist of three parts: *mutation*, *feedback guidance*, and *fuzzing strategy*:

- *Mutation:* According to the difference of the target application program and data format, the corresponding test data generation method is chosen and it can use the pre-generated examples, a variation of valid data examples, or dynamically generated ones according to the protocol or file format.
- *Feedback guidance:* The fuzzy test example is executed, and the target program is executed and monitored. The test data that causes the exception of the target program is recorded.
- *Fuzzing strategy:* If an error is detected, the corresponding example is reported and new generated examples that cover new traces are stored in the example pool.

2.2 CycleGAN

Adversarial Example Generator (*AEG*) is an important part of our approach. To improve the stability and security for target DL systems, *AEG* provides effective adversarial examples to detect potential defects. The idea of generating adversarial examples is to add perturbations that people cannot distinguish from the original examples; this is very similar to the idea of GAN [30] generation of examples. GAN's generators G and discriminators D alternately generate adversarial examples that are very similar but not identical to the original examples based on noise data. Considering the difference of datasets of different target DL systems, such as some DL systems with label data and other DL systems may not, we choose *CycleGAN* [21] as the training model of adversarial example generator, since *CycleGAN* does not require the matching of data sets and label information. *CycleGAN* is one of the most effective adversarial generation approaches. The mapping function and loss function of *CycleGAN* are described as follows.

- The goal of *CycleGAN* is to learn the mapping functions between two domains X and Y. There are two mappings G and F in the model. There are two adversarial discriminators Dx and Dy, where Dx aims to distinguish between images $\{x\}$ and translated images $\{F(x)\}$. Dy has a similar definition.
- Like other GANs, the adversarial loss function is used to optimize the mapping function. But during the actual training stage, it is found that the negative log- likelihood objective is not very stable and the loss function is changed to least-squares loss [31].
- Because of the group mapping, it is impossible to train by using the adversarial loss function only. The reason is that the mapping F can map all x to a picture in Y space, consequently, *CycleGAN* puts forward the cycle consistency loss.

Fig. 2 shows an example structure of *CycleGAN* [21]. The purpose of this example is to transform real pictures and Van Gogh style paintings into each other. It does not need pairs of data to guide the adversarial generation, and has a wide range and practicability. Therefore, in this paper, we use *CycleGAN* to train our adversarial example generator, which can effectively generate adversarial examples to test the target DL systems.

2.3 VGG-19 Network Structure

The ability of deep feature recognition and semantic expression extracted by CNN is stronger. Consequently, it has more advantages than traditional image features. The structure of VGG-19 [22] convolution network is shown in Fig. 3. There are 19 layers including 16 convolution layers, i.e., two every Convl1-Convl2, four every Convl3-Convl5, and three full-connection layers, Fc6, Fc7, and Fc8. The

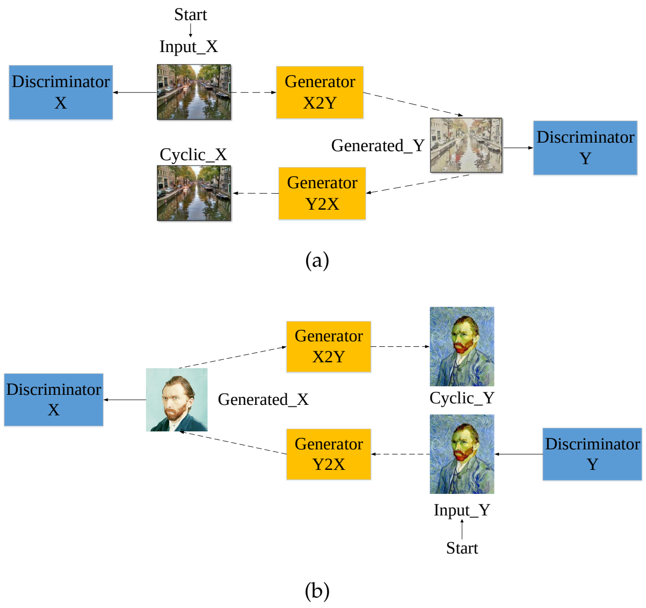


Fig. 2. An example demonstration of CycleGAN. (a) transform the real picture into Van Gogh style painting; (b) transform Van Gogh style painting into the real pictures.

works in [32], [33] show that the VGG-19 network can extract high-level semantic information from images, and it can be used to identify similarities between images. In this paper, the output of the last full connection layer is fused as feature vector to compare the similarity between the adversarial examples and the original examples, and to serve as the threshold for filtering the generated adversarial examples.

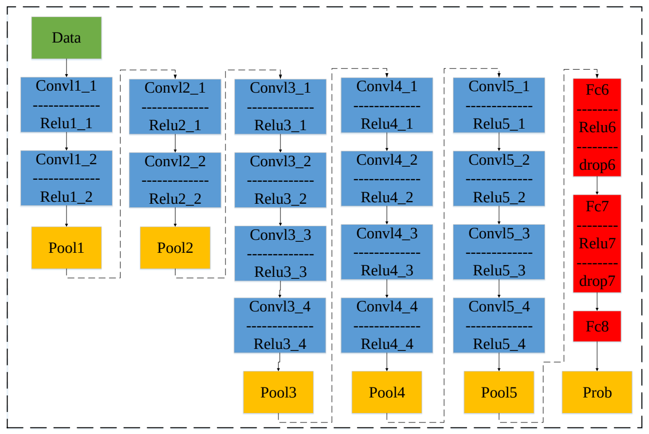


Fig. 3. Structural Chart of VGG19 Network for Extracting Deep Features of Target Images

2.4 Neuron Coverage

Pei et al. [10] propose for the first time neuron coverage as a measure of testing DL. They define neuron coverage of a set of test inputs as the ratio of the number of unique activated neurons in all test inputs to the total number of neurons in the DNN.

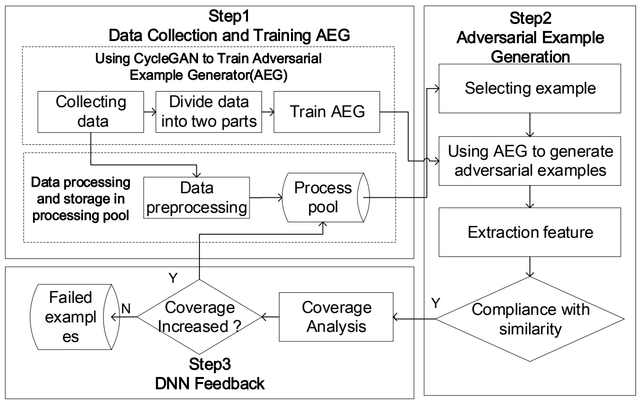

Step1
Data Collection and Training AEG
Using CycleGAN to Train Adversarial Example Generator(AEG)
Collecting data
Divide data into two parts
Train AEG
Data processing and storage in processing pool
Data preprocessing
Process pool
Step2
Adversarial Example Generation
Selecting example
Using AEG to generate adversarial examples
Extraction feature
Compliance with similarity
Y
Coverage Analysis
Coverage Increased ?
Y
N
Failed exampl es
Step3
DNN Feedback

Fig. 4. Coverage-Guided Adversarial Generative Fuzzing Testing Approach

Let $N = \{n_1, n_2, ..., n_p\}$ be the set of all neurons in the DNN, where p is the length of neurons. The input to a DNN is an image $x_i \in T = \{x_1, x_2, ..., x_q\}$, where T is the input domain and q is the length of the input domain. Let $out(n_i, x_i)$ be an output function that returns the output value of a neuron n_i in DNN for a given test input x_i. Finally, let t represent the threshold for considering a neuron to be activated. Then, the neuron coverage can be defined in the following:

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

3 Coverage-Guided Adversarial Generative Fuzzing Testing Approach

In this section, we first give an overview of our approach (Section 3.1), and then we describe the pre-treatment of our approach in Section 3.2, including data collection and *AEG* training. Section 3.3 describes the algorithm of the adversarial example generation process. Finally, Section 3.4 shows how our approach uses neuron coverage feedback to guide the generation of new adversarial examples.

3.1 Overview

The core component of DL systems is the Deep Neural Network (DNN) with different structures and parameters. In the following discussions, we will study how to test DNNs. The input formats of DNNs can be various. In this paper, we focus on DNNs that take pictures as input. Adding perturbations to images has a great impact on DNNs and may cause errors. Guided by neuron coverage, the quality of the generated adversarial examples can be improved. As anticipated before, this paper presents *CAGFuzz*, a coverage-guided adversarial generative fuzzing testing approach. This approach generates adversarial examples with invisible perturbations based on *AEG*. In general, Fig. 4 shows the main process of our approach, which consists of three parts, described as follows:

- The *first step* is the data collection and training adversarial example generator. For each data set, the data sets are divided into two subsets and as the input of *CycleGAN* to train *AEG*. These examples are then put into the processing pool after priority is set according to storage time. We use this processing pool as the initial input for fuzzy testing.

- The *second step* is the adversarial example generation. Each time a prioritized raw example is selected from the processing pool and used as the input of *AEG* to generate adversarial examples. Using deep feature constraint to determine which adversarial examples should be saved. First, we use the VGG-19 network to extract the deep features (see Section 3.3.2) of the original and adversarial examples. Then, we calculate the cosine similarity (see Section 3.3.3) between the deep features of the original and the adversarial examples. If the cosine similarity between the two deep features is more than 0.9, we assume that the adversarial example is consistent with the original example in deep semantics and can be saved.
- The *third step* is to use neuron coverage to guide the generation process. The adversarial examples generated in the second step is given as input to the DNN under test for coverage analysis. If a new coverage occurs, the adversarial example will be put into the processing pool as part of the dataset. The new coverage means that the neuron coverage of the adversarial example is higher than the neuron coverage of the original example.

The main flow chart of the *CAGFuzz* approach is shown in Algorithm 1. The input of *CAGFuzz* includes a target Dataset (D), a deep neural network DNN (DNN), the number of maximum iterations N, the number of adversarial examples $N1$ generated by each original example, and the parameter K of top-k. The output is the generated test example that improves the coverage of the target DNN.

Before the whole fuzzing process, we need to process the dataset. On the one hand, the dataset is divided into two equal data fields (Line 1) to train adversarial example generator *AEG*(Line 2). On the other hand, all examples are pre-processed (Line 3) and stored in the processing pool (Line 4). During each iteration process (Line 5), the original example $parent$ is selected from the processing pool according to the time priority (Lines 6- 7). Then, each original example $parent$ is generated many times (Line 8). For each generation, the adversarial example generator *AEG* is used to mutate the original example $parent$ to generate the adversarial example $data$ (Line 9). The deep features of the original example $parent$ and the adversarial example $data$ are extracted separately, and the cosine similarity (Lines 10- 11) between them is calculated. Finally, all the adversarial examples generated by original example are sorted from high to low in similarity, and top-k of them are selected as the target examples (Line 13). Calculating the neuron coverage of the top-k adversarial examples and feedback these coverage to determine whether the adversarial example is saved (Line 15). If the adversarial examples increase the coverage of the target DNN, they will be stored in the processing pool and set a time priority (Lines 16- 19). The content of time priority is in Section 3.3.1.

Algorithm 1 A description of the main loop of *CAGFuzz*

```
Input: $D$: Corresponding data sets,
       $DNN$: Target Deep Neural Network,
       $N$: The number of maximum iteration,
       $N1$: Number of new examples generated,
       $K$: Top-k parameter
Output: Test Example Set for Increasing Coverage
 1: $X, Y$ = Divide($D$);
 2: Train $AEG$ through $X$ and $Y$
 3: $T$ = Preprocessing($D$);
 4: The preprocessed dataset $T$ serves as the initial processing pool
 5: while $number\ of\ iterations < N$ do
 6:   $S$ = HeuristicSelect($T$);
 7:   $parent$ = Sample($S$);
 8:   while $number\ of\ generation < N1$ do
 9:     $data$ = AEG($parent$);
10:     $Fp, Fd$ = FeatureExtraction($parent$,$data$);
11:     $Similarity$ = CosineSimilarity($Fp$,$Fd$);
12:   end while
13:   Selecting top-k examples from all new examples;
14:   while $number\ of\ calculation < K$ do
15:     $cov$ = DNNFeed($data$);
16:     if IsNewCoverage($cov$) then
17:       Add $data$ to processing pool
18:       Setting time priority for $data$;
19:     end if
20:   end while
21: end while
22: Output all examples in the processing pool as a test example set;
```

3.2 Data Collection and Training AEG

3.2.1 Data Collection

We define the target task of *CAGFuzz* as an image classification problem. Image classification is the core module of most existing DL systems. The first step of *CAGFuzz* is to choose the image classification DNN (e.g. LeNet-1, 4, 5) to be tested and the dataset to be classified. The operation of the dataset is divided into two parts. First, all the examples in the dataset are prioritized, and then all the examples are stored in the processing pool as the original example. During the process of fuzzing, the fuzzer selects the original example from the processing pool according to the priority to perform the fuzzing operation. Second, the dataset is divided into two uniform groups. According to the domain, it is used as the input of the cycle generative adversarial network to train the adversarial example generator.

3.2.2 Training Adversarial Example Generator

Traditional fuzzers mutate the original examples by flipping bits/bytes, cross-input files and swap blocks to achieve the effect of fuzziness. However, mutation of DNN input using these methods is not achievable or invalid, and may produce a large number of invalid and/or non-semantic testing examples. At the same time, how to grasp the degree of mutation is also a question for us to think about. If mutation changes very little, the newly generated examples may be almost unchanged. Although this may be meaningful, the

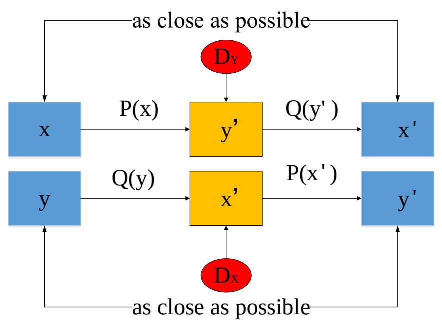


Fig. 5. Transformation relationship between two mapping functions in training *AEG*.

possibility of new examples finding DNN errors is very low. On the other hand, if the mutation changes greatly, more defects of DNN may be found. However, the semantics gap between the new generated example and the original example may be also large, that is to say, the new generated example is also invalid.

We propose a new strategy that uses adversarial example generator as mutations. Given an image example x, *AEG* generates an adversarial example x', and the deep semantics information of x' is consistent with that of x, but the adversarial perturbations that cannot be observed by human eyes are added. We invert the idea of *CycleGAN*, add adversarial perturbations to the original example by adversarial loss, and control the perturbations to be invisible to human eyes by cyclic consistency loss.

In Section 3.2.1, we propose to divide the collected data into two groups of data domains evenly. We define these two data domains as data domain X and data domain Y. Our goal is to use the two data domains as input of the *CycleGAN*, and to learn mapping functions from each other between the two data domains to train the *AEG*. Supposing that the set of data domain X is represented as $\{x_1, x_2, ..., x_n\}$, where x_i denotes a training example in data domain X. Similarly, the set of data domain Y denotes $\{y_1, y_2, ..., y_m\}$, where y_i represents a training example in data domain Y. We define the data distribution of two groups of data domains, where the data domain X is expressed as $x \sim P_{data}(x)$, and data domain Y is expressed as $y \sim P_{data}(y)$. As shown in Fig. 5, the mapping functions between two sets of data domains are defined as $P : X \to Y$ and $Q : Y \to X$, where P represents the transformation from data domain X to data domain Y, and Q represents the transformation from data domain Y to data domain X. In addition, there are two adversarial discriminators D_X and D_Y. D_X distinguishes the original example x of data domain X from the one generated by mapping function Q. Similarly, D_Y distinguishes the original example y of data domain Y from the adversarial example $P(x)$ generated by mapping function P.

Adversarial Loss. The mapping function between two sets of data domains is designed with loss function. For mapping function P and corresponding adversarial discriminator D_Y, the objective function is defined as follows:

$$\begin{aligned}\min_{P}\max_{D} YV(P, D_Y, X, Y) = E_{y\sim P_{data}(y)}[logD_Y(y)]+ \\ E_{x\sim P_{data}(x)}[log(1 - D_Y(P(x)))]\end{aligned} \tag{2}$$

The function of mapping function P is to generate adversarial examples $y' = P(x)$ similar to data domain Y, which can be understood as adding large perturbations with Y characteristics of data domain to the original example x of data domain X. At the same time, there is an adversarial discriminator D_Y to distinguish the real examples y in data domain Y and the generated adversarial example Y'. The objective of the objective function is to minimize the mapping function P and maximize the adversarial discriminator D_Y. Similarly, for the mapping function Q and the target function set by the adversarial discriminator D_X, the objective function is defined in the following:

$$\begin{aligned}\min_{Q}\max_{D} XV(Q, D_X, Y, X) = E_{x\sim P_{data}(x)}[logD_X(x)]+ \\ E_{y\sim P_{data}(y)}[log(1 - D_X(Q(y)))]\end{aligned} \tag{3}$$

Cycle Consistency Loss. We can add perturbations to the original example by using the aforementioned adversarial loss function, but the degree of mutation of this perturbation is large, and it is prone to generate invalid adversarial examples. To avoid this problem, we add constraints to the perturbations, and control the degree of mutation through the cycle consistency loss. In this way, the perturbation-resistant human eyes added to the original example are invisible. For example, example x of data domain X is generated by mapping function P to generate adversarial example y', and then adversarial example y' is generated by mapping function Q to generate new adversarial example x'. At this time, the generated adversarial example x' is similar to the original example x, that is to say, $x \to P(x) = y' \to Q(y') = x' \approx x$. The objective function of the loss function of cyclic consistency is described as follows:

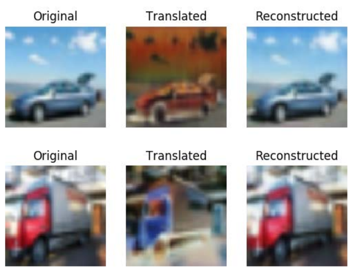
$$\begin{aligned}Loss_{cycle}(P, Q) = E_{x\sim P_{data}(x)}[||Q(P(x) - x||_1]+ \\ E_{y\sim P_{data}(y)}[||P(Q(y) - y||_1]\end{aligned} \tag{4}$$

The overall structure of the network has two generators: P and Q, and two discriminator networks D_X and D_Y. The whole network is a dual structure. We combine two generators with opposite functions into our adversarial example generator. The effect picture of *AEG* is shown in Fig. 6, we show that the adversarial example generation process has 12 groups of pictures of different categories. In each picture, the leftmost column is the original example, the middle column is the transformed example of the original example, and the rightmost column is the reconstructed example. We choose the reconstructed example as the adversarial example. First, larger perturbations are added to the original example. Second, the degree of mutation is controlled by reverse reconstruction to generate adversarial examples with smaller perturbations.

3.3 Adversarial Example Generation

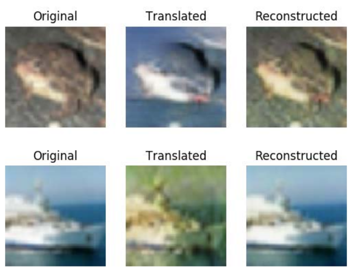
3.3.1 *Example Priority*

The priority of the example determines which kind of examples should be selected next time. We choose a probabilistic

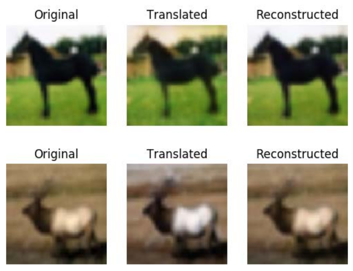


(a) automobile and truck

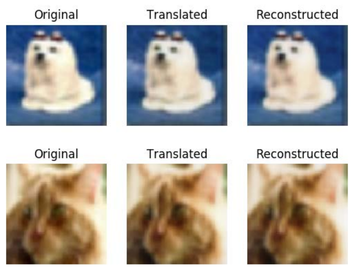
(b) airplane and bird



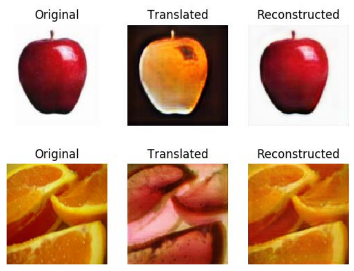
(c) frog and ship



(d) horse and deer



(e) dog and cat



(f) apple and orange

Fig. 6. *AEG* generates effect maps of adversarial examples. In each picture, the leftmost column is the original example, the middle column is the transformed example of the original example, and the rightmost column is the reconstructed example.

selection strategy based on the time of adding examples to the processing pool. We adopt a meta-heuristic formula with faster selection speed. The probability calculation formula is described as follows: $h(b_i, t) = \frac{e^{t_i - t}}{\sum e^{t_i - t}}$, where $h(b_i, t)$ represents the probability of selecting example b_i at time t, and t_i represents the time when example b_i joins the processing pool.

This priority can be understood as follows: the most recently sampled examples are more likely to generate useful new neuron coverage when mutating to adversarial examples. However, when time passes, the advantage will gradually diminish.

3.3.2 Deep Feature

To ensure the meaning of the generated adversarial examples as much as possible, we adopt the strategy of extracting the semantics features of the original examples and adversarial examples and controlling their differences within a certain range. The deep feature recognition ability and semantics expression ability extracted by CNN are stronger. Consequently, we select the VGG-19 network to extract the deep features of examples. The deep features in the VGG-19 model are extracted according to the hierarchy. Compared with the high-level features, the low-level features are unlikely to contain rich semantics information.

The deep features extracted from the VGG-19 network model can represent images better than traditional image features. It also shows that the deeper the layer of convolution network, the more parameters in the network, and the better the image can be expressed. We fuse the output of the last full connection layer (Fc8 layer in Fig. 3) as deep feature, and the dimension of deep feature is 4096.

3.3.3 Cosine Similarity Computation

During the mutation process, *AEG* generates multiple adversarial examples for each original example. We assume that the original example is a, and the set of all adversarial examples is $T = \{a_1, a_2, ..., a_n\}$, which extracts the semantics feature vectors for the original example and all the confrontational examples by the feature extraction method mentioned above. The dimension of each feature vector is 4096. Supposing that the feature vector corresponding to the original example a is $X = [x_1, x_2, ..., x_n]_{n=4096}$, and the corresponding eigenvector of an adversarial example is a_i is $Y = [y_1, y_2, ..., y_n]_{n=4096}$, where $a_i \in T$. Cosine similarity is used to measure the difference between each adversarial example and the original example. The formula is described as follows:

$$COS(X, Y) = \frac{X \cdot Y}{||X|| \times ||Y||} = \frac{\sum_{i=1}^{n}(x_i \times y_i)}{\sqrt{\sum_{i=1}^{n} x_i^2} \times \sqrt{\sum_{i=1}^{n} y_i^2}} \tag{5}$$

where x_i and y_i correspond to each dimension of eigenvector X and Y.

To control the size and improve the mutation quality of adversarial examples, we select the top-k adversarial examples sorted from high to low cosine similarity as eligible examples to continue the follow-up steps. In our approach, we set $K = 5$, that is to say, we select the five adversarial examples with the highest cosine similarity for neuron coverage.

3.4 DNN Feedback

Without coverage as a guiding condition, the adversarial examples generated by *AEG* are not purposeful. Consequently it is impossible to know whether the adversarial examples are effective or not. If the generated adversarial examples cannot bring new coverage information to the DNN to be tested, these adversarial examples can only simply expand the dataset, but cannot effectively detect the potential defects of DNN. To make matters worse, mutations in these adversarial examples may bury other meaningful examples in a fuzzy queue, thus significantly reducing the fuzzing effect. Therefore, neuron coverage feedback is used to determine whether the newly generated adversarial examples should be placed in the processing pool for further mutation.

After each round of generation and similarity screening, all valid adversarial examples are used as the input of DNN to be tested for neuron coverage analysis. If the adversarial examples generate new neuron coverage information, we will set priority for the adversarial examples and store it in the processing pool for further mutation. For example, a DNN for image classification consists of 100 neurons. 32 neurons are activated when the original example is input into the network, and 35 neurons are activated when the adversarial example is input into the network. Consequently, we say that the adversarial example brings new coverage information.

4 Experimental Evaluation

In this section, we perform a set of dedicated experiments to validate *CAGFuzz*. Section 4.1 proposes the research questions. Section 4.2 describes the experimental design. Section 4.3 provides the experimental results and Section 4.4 discusses some threats to validity.

4.1 Research Questions

We use three standard deep learning datasets and the corresponding image classification models to carry out a series of experiments to validate *CAGFuzz*. The purpose of the experiments is designed to explore the following four main research questions:

- *RQ1*: Could the generated adversarial examples based on data have stronger generalization ability than those based on models?
- *RQ2*: Could *CAGFuzz* improve the neuron coverage in the target network?
- *RQ3*: Could *CAGFuzz* find potential defects in the target network?
- *RQ4*: Could the accuracy and the robustness of the target network be improved by adding adversarial examples to the training set?

To discover potential defects of target network and expand effective examples for data sets, the *CAGFuzz* approach mainly generates adversarial examples for DNNs to be tested. Therefore, we designed *RQ1* to explore whether the examples generated based on data have better generalization ability than those based on models. For neuron coverage, we designed *RQ2* to explore whether *CAGFuzz* can effectively generate test examples with more coverage information for target DNNs. We designed *RQ3* to study whether *CAGFuzz* can discover more hidden defects in target DNNs. *RQ4* is designed to explore whether adding the adversarial examples generated by *CAGFuzz* to the training set can significantly improve the accuracy of target DNNs.

4.2 Experimental Design

4.2.1 Experimental Environment

The experiments have been performed on Linux machines. The detailed descriptions of the hardware and software environments of the experiments are shown in Table 1.

TABLE 1
Experimental hardware and software environment

Name	Standard
CPU	Xeon Silver 4108
GPU	NVIDIA Quadro P4000
RAM	32G
System	Ubuntu 16.04
Programming environment	Python
Deep learning open source framework	Tensorflow1.12

4.2.2 DataSets and Corresponding DNN Models

For research purpose, we adopt three popular and commonly used datasets with different types of data: MNIST [23], CIFAR-10 [24], and ImageNet [25]. At the same time, we have learned and trained several popular DNN models for each dataset, which have been widely used by scientific researchers. In Table 2, we provide an informative summary of these datasets and the corresponding DNN models. All the evaluated DNN models are either pre-trained (i.e., we use the common weights in previous researchers' papers) or trained according to standards by using common datasets and public network structures.

MNIST [23] is a large handwritten digital dataset containing $28 * 28 * 1$ pixels of images with class labels ranging from 0 to 9. The dataset contains 60,000 training examples and 10,000 test examples. We construct three different kinds of neural networks based on LeNet family, namely LeNet-1, LeNet-4, and LeNet-5.

CIFAR-10 [24] is a set of general image classification images, including $32 * 32 * 3$ pixel three-channel images, including ten different kinds of pictures (such as aircraft, cats, trucks, etc.). The dataset contains 50,000 training examples and 10,000 test examples. Due to the large amount of data and high complexity of CIFAR-10, its classification task is more difficult than MNIST. To obtain the competitive performance of CIFAR-10, we choose three famous DNN models VGG-16, VGG-19, and ResNet-20 as the targeted models.

ImageNet [25] is a large image dataset, in which each image is a $224 * 224 * 3$ three-channel image, containing 1000 different types. The dataset contains a large number of training data (more than one million) and test data (about 50,000). Therefore, for any automated testing tool, working on ImageNet-sized datasets and DNN models is a severe test. Because the large number of images in the ImageNet dataset, most state-of-the-art adversarial approaches are only evaluated on a part of the ImageNet dataset. To obtain the competitive performance of ImageNet, we choose three famous DNN models VGG-16, VGG-19, and ResNet-50 as the targeted models.

TABLE 2
Subject datasets and DNN models

DataSet	DataSet Description	Model	#Layer	#Neuron	Test acc(%)
MNIST	Hand written digits from 0 to 9	LeNet-1	7	52	98.25
		LeNet-4	8	148	98.75
		LeNet-5	9	268	98.63
CIFAR-10	10 class general image	VGG-16	16	19540	86.84
		VGG-19	19	41118	77.26
		ResNet-20	70	4861	82.86
ImageNet	1000-class large scale datasets	VGG-16	16	14888	92.6
		VGG-19	19	16168	92.7
		ResNet-50	176	94059	96.43

4.2.3 Contrast Approaches

As surveyed in [34], there are several open-source tools in testing machine learning applications. Some released tools, such as *Themis* [2], *mltest* [3], and *torchtes* [4] do not focus on generating adversarial examples. Thus, to measure the ability of *CAGFuzz*, we selected the following three representative DL testing approaches proposed recently in the literature as our contrast approaches, respectively:

- *FGSM* [18] (Fast Gradient Sign Method) - a typical approach generates adversarial examples based on model. Consequently, we use *FGSM* to generate adversarial examples to compare with *CAGFuzz*, and verify that the generated adversarial examples based on pure data have higher generalization ability than those based on models.
- *DeepHunter* [11] - an automated fuzz testing framework for hunting potential defects of general-purpose DNNs. *DeepHunter* performs metamorphic mutation to generate new semantically preserved tests, and leverages multiple plug-able coverage criteria as feedback to guide the test generation from different perspectives.
- *DeepXplore* [10] - the first white box system for systematically testing DL systems and automatically identify erroneous behaviors without manual labels. *DeepXplore* performs gradient ascent to solve a joint optimization problem that maximizes both neuron coverage and the number of potentially erroneous behaviors.

Since there is no open source version of *DeepHunter* [11], we have implemented eight image transformation methods mentioned in *DeepHunter*, and we use these eight methods to replace *DeepHunter* for later experimental evaluation. The source code of *FGSM* and *DeepXplore* can be found on GitHub, and the tools are utilized for later experimental evaluation.

2. http://fairness.cs.umass.edu/
3. https://github.com/Thenerdstation/mltest
4. https://github.com/suriyadeepan/torchtest

4.3 Experimental Results

4.3.1 Training of Target DNNs

To ensure the correctness and validate the evaluation results of the experiments, we carefully select several popular DNN models with competitive performance for each dataset. These DNN models have been proven to be standard in previous researchers' experiments. In our approach, we closely follow the common machine learning training practices and guidelines, and set the learning rate for training DNN model. During the initialization process of DNN model learning rate, if the learning rate is too high, the weight of the model will increase rapidly, which will have a negative impact on the training of the whole model. Consequently, the learning rate is set to a smaller value at the beginning. For the three LeNet models of the MNIST dataset, we set the learning rate as 0.05.

For the two VGG networks of the CIFAR-10 dataset, we set the initial learning rate as 0.0005 based on experiences because of the deeper network layers and the more complex model. In addition, we initially set the epoch for each model as 100 training times. The LeNet model works well, but when we train the VGG-16 network, we find that the accuracy of the model is basically stable after 50 training epochs, as shown in Fig. 7. Therefore, during the process of training the VGG-19 network and the subsequent retraining model stage, we reset the training epochs to 50; this can save a lot of computing resources and space-time costs. During the process of training the ResNet-20 model, we set up a three-stage adaptive learning rate. When $epoch < 20$, we set the learning rate as $1e^{-3}$. When $20 < epoch < 50$, we set the learning rate as $1e^{-4}$. When $epoch > 50$, we set the learning rate as $1e^{-5}$.

For the VGG-16, VGG-19, and ResNet-50 models used to classify the ImageNet data sets, we directly used the model with ImageNet as the data set in the keras framework [35], since it has already trained and achieved enough performance. The Imagenet data set is too large (including 137 GB training set, 12.7 GB test set, and 6.28 GB verification set), the cost of retraining model is too large, and the general hardware equipment cannot meet the requirements. Therefore, for the ImageNet data set, we only have performed experiments on the two modules (Neuron Coverage and Error Behavior Discovery).

In Fig. 8, we show the training loss, training accuracy, validation loss, and validation accuracy of each model. As can be seen in the figure, during the training process of LeNet-5, with the increase of training times, the loss value of the model gradually decreases, and the accuracy is getting higher and higher. This shows that with the increase of training time, the model can fit the data well, and the model can accurately classify the dataset. We follow the criterion of machine learning, and then choose the competitive DNN models as the research object for the fuzzy test under the condition of fitting.

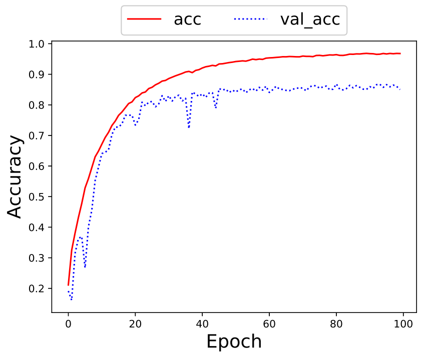


Fig. 7. Sketch of VGG-16 network training accuracy and verification accuracy change with training epoch

4.3.2 Generalization Ability

To answer *RQ1*, we compare *CAGFuzz* with the existing model-based approach *FGSM*. In the experiment, we enhanced *FGSM* by adding the coverage feedback to the generated adversarial examples. In this way, *FGSM* has the same coverage-guided test approach as *CAGFuzz*. We choose MNIST and CIFAR-10 data sets as the sampling set. For MNIST, we sample 10 examples for each class in the training set and 4 examples for each class in the test set. Since the DNN models used to classify CIFAR-10 data set have a large scale of weight parameters, 10 training examples are not enough to achieve the training effect. Therefore, for CIFAR-10, we sample 100 examples for each class in the training set and 10 examples for each class in the test set.

Based on the LeNet-1 model, we use *FGSM* to generate an adversarial example for each of the 10 examples in the training set, and also use *AEG* to generate an adversarial example for each training example. First, the original data set is used to train and test the LeNet-1 model. We set the epoch as 50 and the learning rate as 0.05. Then, the adversarial examples generated by *CAGFuzz* and *FGSM* are added to the training set to retrain LeNet-1 with the same parameters. Finally, the above two steps are repeated, but the model is replaced by LeNet-4 or LeNet-5.

Similar to generating adversarial examples based on LeNet-1, we perform the same experiment on LeNet-4 and LeNet-5. Because of the uncertainty during the model training process, we train the model repeatedly 5 times in the same setting, and take the average of these results as the final accuracy of our experiments. For example, the 5 times accuracy of the ResNet-20 model under FGSM-R20 fluctuates; therefore, we take the average value. Table 3 shows the accuracy of the three models on the original data set, FGSM-Le1, FGSM-Le4, FGSM-Le5 and CAGFuzz-dataset. Among them, “FGSM-Le1” refers to the data set generated by *FGSM* method. ”CAGFuzz-dataset” refers to the data set generated based on *CAGFuzz*. From the table, it can be seen that the accuracy of the adversarial examples generated by *FGSM* based on a specific model is improved higher than that of other models. For example, after retraining LeNet-1 based on FGSM-Le1, the accuracy is 70.6%. After retraining LeNet-1 based on FGSM-Le4 and FGSM-Le5, the accuracy is 66.6% and 68.6%. Analyzing all data in the Table 3, we can see that after retraining three models based on CAGFuzz-dataset, the accuracy of the models are all high, namely, 72.6%, 72% and 74.3%. In the same way, similar results are obtained when applied to the CIFAR10 data set, and the final results are shown in Table 4. We can see that after retraining three models based on CAGFuzz-dataset, the accuracy of the model is mostly higher than the maximum accuracy of the retraining model based on *FGSM*. In the ResNet-20 model, the final accuracy of *CAGFuzz* retraining model is 39.2%, which is a little worse than FGSM-R20, but much better than FGSM-V16 and FGSM-V19.

▷ *Answer to* ***RQ1***: Taking the MNIST and CIFAR-10 data sets as examples, we prove that the adversarial examples generated based on target model (such as *FGSM*) only improve the accuracy of this special model better, and the improvement on other models is limited. On the contrary, *CAGFuzz* can generate adversarial examples based on data, and this can improve the accuracy of all the models to almost the same degree. Summarizing, the adversarial examples generated based on *CAGFuzz* has better generation ability of the adversarial examples generated based on target model.

TABLE 3
The accuracy of the three models on the MNIST data set, adversarial examples generated based on *FGSM* and adversarial examples generated based on *CAGFuzz*(%)

Model	Orig. dataset	FGSM-Le1	FGSM-Le4	FGSM-Le5	CAGFuzz-dataset
LeNet1	59	**70.6**	66.6	68.6	**72.6**
LeNet4	62.6	66.6	**71.6**	68.2	**72**
LeNet5	60.6	69.3	64.6	**71**	**74.3**

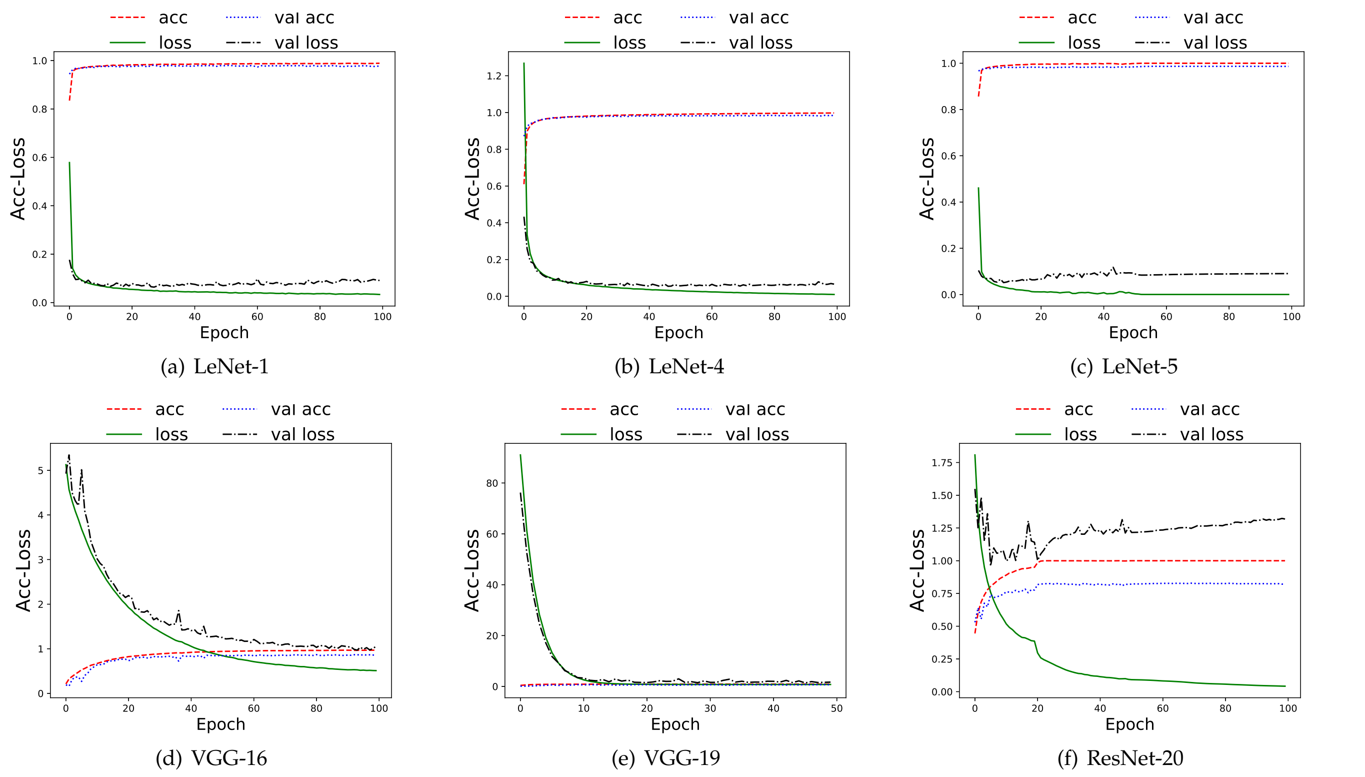
TABLE 4
The accuracy of the three models on the CIFAR10 data set, adversarial examples generated based on *FGSM* and adversarial examples generated based on *CAGFuzz*(%)

Model	Orig. dataset	FGSM-V16	FGSM-V19	FGSM-R20	CAGFuzz-dataset
VGG16	19	**28.2**	21.8	24	**30.2**
VGG19	10	18.4	**25.6**	21.4	**27**
ResNet20	15	33.8	36.8	**40**	**39.2**

4.3.3 Neuron Coverage

To answer *RQ2*, we use the training data set of each model as the input set to calculate the original neural coverage, and the generated adversarial example set as the input set to calculate the neural coverage of *CAGFuzz*.

Obviously, the adversarial examples generated by *AEG* can effectively improve the neuron coverage of the target DNNs. To further validate the effectiveness of *CAGFuzz* in improving neuron coverage, we also compare it with other three approaches. Table 5 lists the original neuron coverage of each model and the neuron coverage using the different approaches. It can be seen from the table that in the MNIST data set, the *FGSM* approach does not improve the coverage of the model. For the LeNet-1 and



(a) LeNet-1 (b) LeNet-4 (c) LeNet-5

(d) VGG-16 (e) VGG-19 (f) ResNet-20

Fig. 8. Model training record diagram:(a) LeNet-1, training on MINIST Data Set, when epoch=100, (b) LeNet-4, training on MINIST Data Set, when epoch=100, (c)LeNet-5, training on MINIST Data Set, when epoch=100, (d)VGG-16, training on CIFAR-10 Data Set, when epoch=100, (e)VGG-19, training on CIFAR-10 Data Set, when epoch=50, (f)ResNet-20, training on CIFAR-10 Data Set, when epoch=100.

LeNet-4 models, the coverage improvement of *CAGFuzz* is not better than *DeepHunter* and *DeepXplore*. However, the coverage improvement effect of *CAGFuzz* in the LeNet-5 model is obvious better than the other two approaches. In the CIFAR-10 data set, the coverage improvement of the *FGSM* approach is also not good, and even worse than the coverage of the original examples. The coverage improvement of *CAGFuzz* is generally better than other approaches, in addition to the ResNet-20 model, *DeepHunter* increases the coverage to 78.62%, while *CAGFuzz* only increases to 75.74%. In the ImageNet data set, *CAGFuzz* can improve the model coverage better than all other approaches.

▷ *Answer to* ***RQ2****:* In conclusion, *CAGFuzz* can effectively generate adversarial examples and these adversarial examples can improve neuron coverage for the target DNN. Due to the deep feature constraint, the adversarial examples generated by *CAGFuzz* can significantly improve the neuron coverage in the model with deep depth and large number of neurons.

TABLE 5
Comparison of *CAGFuzz*, *FGSM* [18], *DeepHunter* [11] and *DeepXplore* [10] in Increasing the Neuron Coverage of Target DNNs.

DNN Model	Orig. NC(%)	*FGSM* NC(%)	*DeepHunter* NC(%)	*DeepXplore* NC(%)	*CAGFuzz* NC(%)
LeNet1	38.46	38.46	53.84	57.69	**46.15**
LeNet4	72.41	72.41	80.17	81.89	**79.31**
LeNet5	86.44	86.44	88.98	87.28	**93.22**
VGG16	50.99	47.30	59.71	55.39	**62.32**
VGG19	55.33	55.47	57.34	56.02	**58.51**
ResNet20	75.04	75.33	78.62	75.37	**75.74**
VGG16	13.33	13.91	14.07	13.68	**14.54**
VGG19	13.98	14.74	15.24	14.01	**16.36**
ResNet50	76.88	77.28	76.44	77.96	**78.26**

4.3.4 Error Behavior Discovery

To answer *RQ3*, we sample correctly classified examples by DNN models from the test set of each dataset. Based on these correctly classified examples, we generated adversarial examples for each example through the *AEG* of each dataset. The examples we selected are all positive examples with correct classification. We can confirm that all the generated adversarial examples should also be classified correctly, because the deep semantics information of the adversarial examples and the original examples are consistent. The "positive examples" generated by *AEG* are input into the corresponding classifier model for classification. If there are errors or classification errors, a potential defect of the classification model can be found. We define the original correct example as $Image_{orig}$ and the corresponding adversarial example as $Image_{adv} = \{Image_1, Image_2, ..., Image_{10}\}$. The original example $Image_{orig}$ is classified correctly in the target DNN model, consequently $Image_i$ should also be classified correctly, where $Image_i \in Image_{adv}$. If the $Image_i$ classification of an adversarial example is wrong, we consider this to be an error behavior of the target DNN.

We choose a quantitative measure to evaluate the ef-

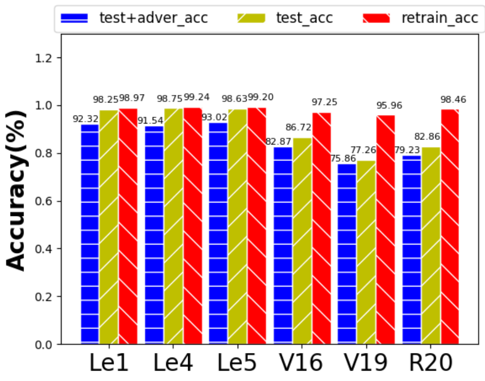


Fig. 9. Improvement of accuracy and robustness after retraining model.

fectiveness of *CAGFuzz* in detecting erroneous behaviors in different models. As mentioned above, we take 2000 examples, which are verified to be correct from each data set. Then we use the four approaches mentioned to mutation these examples, and generate 2000 adversarial examples for our experiments. Table 6 shows the number of erroneous behaviors found by different datasets under the guidance of neuron coverage. In addition, we also list the number of errors found by *FGSM* [18], *DeepHunter* [11], and *DeepXplore* [10] in each data set.

TABLE 6
Number of erroneous behaviors reported by *FGSM* [18], *DeepHunter* [11], *DeepXplore* [10], and *CAGFuzz* across 2000 adversarial examples.

Data Sets	*FGSM*	*DeepHunter*	*DeepXplore*	*CAGFuzz*
MNIST	162	670	34	**894**
CIFAR-10	69	193	20	**284**
ImageNet	278	456	18	**720**
SUM	509	1319	72	**1898**

As can be seen from Table 6, the *DeepXplore's* ability to detect potential errors is poor, and its performance in each data set is not ideal. The total number of potential errors found in the three data sets is 72. Compared with the other three approaches, *CAGFuzz* has a stronger ability on finding potential errors in the model. It has good results in all the three datasets, and a total of 1898 potential errors in the model have been found.

▷ *Answer to* ***RQ3****:* With neuron coverage guided adversarial examples, and based on the same model and the same positive examples, *CAGFuzz* can find more potential errors in the model.

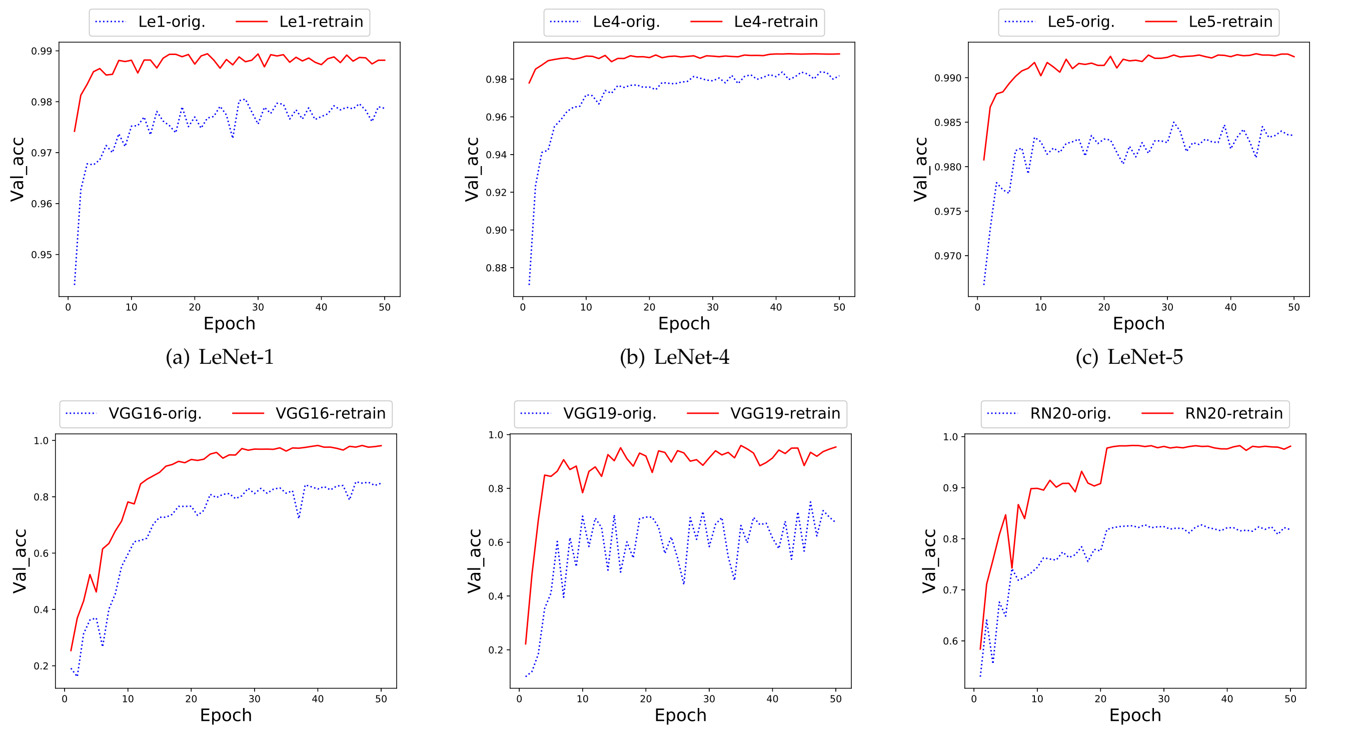
4.3.5 Accuracy and Robustness

To answer *RQ4*, we add adversarial examples generated by *CAGFuzz* to the training set to retrain the DNN model and measure whether it can improve the accuracy of the target DNN. We select the MNIST and CIAR-10 data sets as our experimental data sets, and we select three DNN models, LeNet-1, 4, 5, and the VGG-16, VGG-19, and ResNet-20 models as experimental models. We retrain the DNN model by mixing 65% of the adversarial example set and the original training set, and then validate the DNN model with the remaining 35% of the adversarial example set and the original test set on the original model and the retraining model. Because of the limitation of the size of the picture, in Fig. 9, we abbreviate the model name. For example, the model LeNet-1 is abbreviated to Le1, the model VGG-16 is abbreviated to V16, and the model ResNet20 is abbreviated to R20. In Fig. 9, "test_acc" represents the accuracy of the model on the original test set, "test+adver_acc" represents the accuracy of the model on the test set with adversarial examples (the model is still the original one), and "retrain_acc" represents the accuracy of the model after retraining the model with the adversarial examples. It can be seen from the comparison of "test_acc" and "test+adver_acc" that the robustness of the original model is very poor. After adversarial examples are added into the test set, the accuracy of the model decreases obviously. For example, the accuracy of the LeNet-5 model decreases from 98.63% to 93.02% and from 5.69% on the original basis. In the VGG-19 model, the accuracy of the model decreases from 77.26% to 75.86%. The comparison of "test_acc" and "retrain_acc" shows that the accuracy of the models has been greatly improved after retraining the model with the adversarial examples, especially for the VGG model with deeper layers. For example, from Fig. 9, we can see that the accuracy of the VGG-19 network has increased from 77.26% to 95.96%, with an increase of 24.2%. In general, we can see that *CAGFuzz* can not only improve the robustness of the model, but also improve the accuracy of the model, especially for the model with deeper network layer.

In the experiments, we further analyze the accuracy of the retraining model and the original model during the training process, and evaluate the validity of the adversarial examples generated by *CAGFuzz* from the change of the validation accuracy. Fig. 10 shows the changes of validation accuracy of different models during training. The original structure parameters and learning rate of each model are kept unchanged, and the new data set we reconstituted is used for retraining. During the training process, the validation accuracy and the original validation accuracy of the same epoch are compared. It can be found that under the same epoch, the validation accuracy of the retraining model is higher than that of the original model, and the convergence speed of the retraining model is faster. Moreover, it can also be found from the figure that the retraining model is more stable and has a smaller change range during the training process.

In addition, we can see that the trend of the retrained model is basically consistent with the original model, which shows that the accuracy of the model can be greatly improved without affecting the internal structure and logic of the model. For example, in Fig. 10(d), the accuracy of the original model drops suddenly when epoch = 6, and the retraining model also continues this change. In Fig. 10(f), the original model presents a three-stage upgrade, which is reflected in the retraining model at the same time.

To further validate our approach, we pre-train models on the MNIST and CIFAR-10 data sets. We further expand the training data by adding the same number of new generated examples, and train DNNs by 5 epochs. Our experiment results shown in Fig. 11 are compared with other approaches.



(a) LeNet-1 (b) LeNet-4 (c) LeNet-5

(d) VGG-16 (e) VGG-19 (f) ResNet-20

Fig. 10. Validation Set Accuracy Contrast Diagram of Each Model in the Training Process: (a) LeNet-1, training on MINIST Data Set, when epoch=50, (b) LeNet-4, training on MINIST Data Set, when epoch=50, (c) LeNet-5, training on MINIST Data Set, when epoch=50, (d) VGG-16, training on CIFAR-10 Data Set, when epoch=50, (e)VGG-19, training on CIFAR-10 Data Set, when epoch=50, (f)ResNet-20, training on CIFAR-10 Data Set, when epoch=70.

It can be found that *CAGFuzz* sometimes has a low initial accuracy when the model is retrained. With the increase of epochs, the accuracy of the model increases rapidly, and the final accuracy for *CAGFuzz* is higher than that of other approaches.

▷ *Answer to **RQ4**:* The accuracy of a DNN can be improved by retraining the DNN with adversarial examples generated by *CAGFuzz*. The accuracy of the best model is improved from the original 86.72% to 97.25%, with an improvement of 12.14%.

4.4 Threats to Validity

In the following, we describe the main threats to validity of our approach in detail.

Internal validity: During the experimental process, the data set used to train *AEG* is manually divided into two data domains, which may lead to subjective differences. To mitigate this threat, after the data domain is divided, we asked three observers to randomly exchange the examples of the two data domains, and three selected observers complete independently. In addition, we pre-train with the initial data domains and then retrain with the data domains adjusted by other observers.

External validity: During the experimental process, the classification of experimental data set is limited, which may lead to the reduction of the generality of the approach to a certain extent. To solve this problem, we use a cross-data set approach to validate the generalization performance of the approach.

Conclusion validity: According to the designed three problems, we can validate our approach. To further ensure the validity of the conclusion, we validated the conclusion through the valid data sets and models from other researchers, and reached the same conclusion as the standard data set.

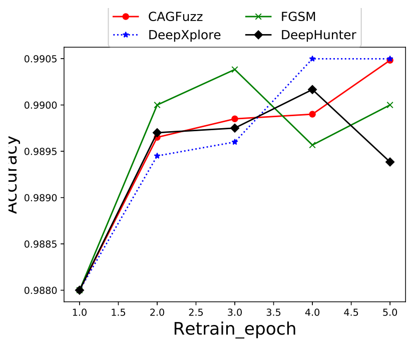
5 Related Work

In this section, we review the most relevant work in three aspects. Section 5.1 introduces the adversarial deep learning and some adversarial examples generation approaches. Section 5.2 elaborates coverage-guided fuzz testing of traditional software. Section 5.3 introduces the state-of-the-art testing approaches of DL systems.

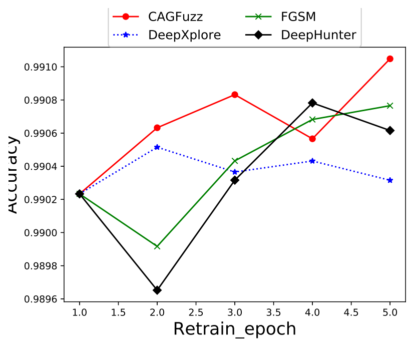
5.1 Adversarial Deep Learning

A large number of recent research has shown that adversarial examples with small perturbations poses a great threat to the security and robustness of DL systems [19], [36], [37], [38]. Small perturbations to the input images can fool the whole DL systems, and the input image is initially classified correctly by the DL systems. Although in human eyes, the modified adversarial example is obviously indistinguishable from the original example.

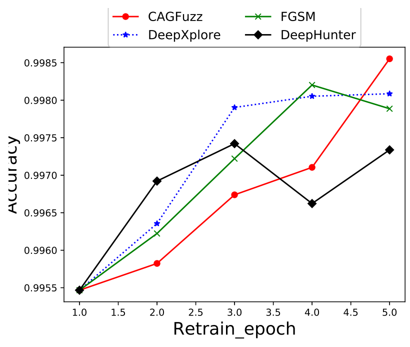
Goodfellow et al. [18] proposed *FGSM* (Fast Gradient Sign Method) which can craft adversarial examples using loss function $J(\theta, x, y)$ with respect to the input feature vector, where θ denotes the model parameters, x is the input,



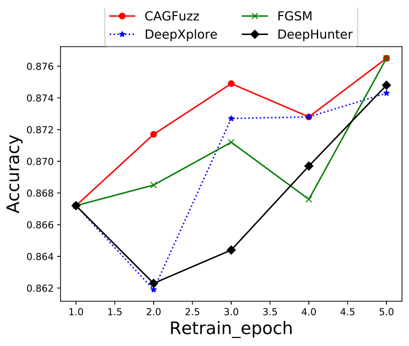
(a) LeNet-1



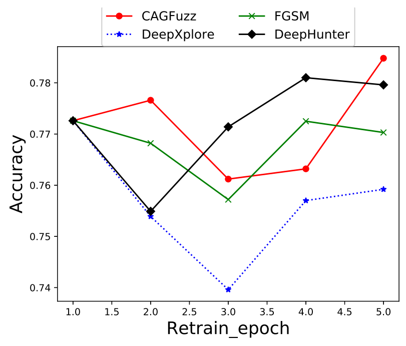
(b) LeNet-4



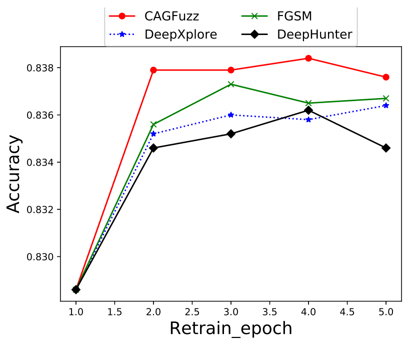
(c) LeNet-5



(d) VGG-16



(e) VGG-19



(f) ResNet-20

Fig. 11. Improvement in accuracy of DNN models when the training set is augmented with the same number of inputs generated by *FGSM*, *DeepXplore*, *DeepHunter* and *CAGFuzz* .

and y is the output label of x. The adversarial example is generated as: $x^{'} = x + \epsilon sign(\nabla_x J(\theta, x, y))$. In this paper, we choose *FGSM* as a baseline. The *FGSM* approach uses the gradient change of specific DNN to generate adversarial examples. Consequently, the generated adversarial examples have good defect detection ability for the specific DNN. However, the approach cannot achieve good performance when it is extended to other DNNs.

Papernot et al. [39] proposed JSMA (Jacobian-based Saliency Map Attack) to craft adversarial examples based on a precise understanding of the mapping between inputs and outputs of DNNs. For an input x and a neural network N, the output of class j is denoted as $N_j(x)$. To achieve a target misclassification class t, $N_t(x)$ is increased while the probabilities $N_j(x)$ of all other classes $j \neq t$ decrease, until $t = \arg max_j N_j(x)$.

Kurakin et al. [40] proposed BIM (Basic Iterative method). They apply it multiple times with small step size, and clip pixel values of intermediate results after each step to ensure that they are in an ϵ-neighbourhood of the original image. The method applies adversarial noise η many times iteratively with a small parameter ϵ, rather than one η with one ϵ at a time, which gives a recursive formula: $x_0^{'} = x$ and $x_i^{'} = clip_{x,\epsilon}(x_{i-1}^{'} + \epsilon sign(\nabla_{x_{i-1}^{'}} J(\theta, x_{i-1}^{'}, y))$, where $clip_{x,\epsilon}(.)$ denotes a clipping of the values of the adversarial example such that they are within an ϵ-neighborhood of the original input x.

Carlini et al. [41] proposed CW (Carlini and Wagner Attacks), a new optimization-based attack technique which is arguably the most effective in terms of the adversarial success rates achieved with minimal perturbation. In principle, the CW attack is to approximate the solution to the following optimization problem: $\arg min_{x^{'}} \lambda L(x, x^{'}) - J(\theta, x^{'}, y)$, where L is a loss function to measure the distance between the prediction and the ground truth, and the constant λ is to balance the two loss contributions.

At present, these approaches are not used for testing deep learning systems. We find that it is meaningful to apply them to the steps of example generation in deep learning test. However, all these approaches only attempt to find a specific kind of error behavior, that is, to force incorrect prediction by adding minimum noise to a given example. In this way, these approaches are designed for special DNNs, and the generated adversarial examples have low generalization ability. In contrast, our approach does not depend on a specific DNN, and uses the distribution of general data domains to learn from each other, so as to add small perturbations to the original examples.

5.2 Coverage-Guided Fuzzing Testing

Coverage-guided fuzzing testing (CGF) [42] is a mature defect and vulnerability detection technology. A typical CGF usually performs the following loops: 1) selecting seeds from the seed pool; 2) mutating seeds for a certain number of times to generate new tests using bit/byte flip, block substitution, and crossover of two seed files; 3) running the target program for the newly generated input and recording the execution trajectory; 4) if the detection is made in example of collapse, the fault seeds are reported and the interesting seeds covered with new traces are stored in the seed pool.

Superion [43] conceptually extends LangFuzz [44] with coverage-guided: the seeds of structural mutation that increase coverage are retained to further fuzzing. While Superion works well for highly structured inputs such as XML and JavaScript, AFLSMART's variation operators better support block based file formats such as image and audio files.

Zest [45] and libprotobuf mutator [46] have been proposed to improve the mutation quality by providing structure aware mutation strategies. Zest compiles the syntax specification into a fuzzer driver stub for the coverage-guided greybox fuzzer. This fuzzer driver translates byte-level mutations of LibFuzzer [27] into structural mutations of the fuzzer target.

NEZHA [47] is used to focus on inputs that are more likely to trigger logic errors by using behavioral asymmetries between test programs. The behavior consistency between different implementations acts as Oracle to detect functional defects.

TensorFuzz [48] is good at automatically discovering errors that only a few examples can cause. For example, it can find the numerical error in the trained neural network, generate the difference between the neural network and its quantized version, and find the bad behavior in the character level language model. However, the defects of *TensorFuzz* are as follows. First, *TensorFuzz* directly adds noise to the examples, so it is unnatural to generate examples, while *CAGFuzz* uses *AEG* to mutate the examples first and then to restore them; thus, in *CAGFuzz* the generated adversarial examples are more natural and understandable for humans. Second, *TensorFuzz* does not consider the deep feature while *CAGFuzz* uses deep feature to constrain the adversarial examples; this enables us to ensure that the high-level semantics of the examples remain unchanged.

The validity of *DLFuzz* [14] shows that it is feasible to apply the fuzzy knowledge to DL testing, which can greatly improve the performance of existing DL testing technologies such as *DeepXplore* [10]. Gradient-based optimization problem solution ensures simple deployment and high efficiency of the framework. The mechanism of seed maintenance provides different directions and more space for improving the coverage of neurons.

Due to the inherent difference between DL systems and traditional software, traditional CGF cannot be directly applied to DL systems. In our approach, CGF is adopted to be suitable for DL systems. The state-of-the-art CGF mainly consists of three parts: mutation, feedback guidance, and fuzzing strategy, in which we replace mutation with the adversarial example generator trained by *CycleGAN*. In the feedback part, neuron coverage is used as the guideline. In the fuzzy strategy part, because the test is basically input by the same format of images, the adversarial examples generated with higher coverage are selected and put into the processing pool to maximize the neuron coverage of the target DL systems.

5.3 Testing of DL Systems

In traditional software testing, the main idea of evaluating machine learning systems and deep learning systems is to randomly extract test examples from manually labeled datasets [49] and hoc simulations [50] to measure their accuracy. In some special cases, such as autopilot, special non-guided simulations are used. However, without understanding the internal mechanism of models, such black-box test paradigms cannot find different situations that may lead to unexpected behavior [10], [51].

DeepXplore [10] proposes a white-box differential testing technique for generating test inputs that may trigger inconsistencies between different DNNs, which may identify incorrect behavior. For the first time, this method introduced concept of neuron coverage as a metric of DL testing. At the same time, it requires multiple DL systems with functions similar to cross-reference prediction to avoid manual checking. However, cross-references have difficulties in finding DL-like systems. *DeepXplore* is similar to *FGSM* in that it is also based on the given DNN model to generate adversarial examples, whose generalization ability is not good. Furthermore, through our experiments, we found that *DeepXplore* has a poor ability on finding potential errors in the model. The reason may be that *DeepXplore* does not use any constraints to control the generation of adversarial examples. In contrast, our approach, *CAGFuzz* uses *AEG* to generate adversarial examples based on a given data set. Experiments show that the generalization ability of these adversarial examples is better. In addition, we use deep feature to constrain the generation of adversarial examples, which has a good effect in finding potential errors of the model.

DeepHunter [11] performs mutations to generate new semantic retention tests, and uses multiple pluggable coverage criteria as feedback to guide test generation from different perspectives. Similar to traditional coverage-guided fuzzy (CGF) testing [52], [53], *DeepHunter* uses random mutations to generate new test examples. Although there is a screening mechanism to filter invalid use examples, it still wastes time and computing resources. *DeepHunter* uses pixel value transformation (change image contrast, image brightness, image blur and image noise) and affine transformation (image translation, image scaling, image shearing, and image rotation) to mutate the image. The examples generated by these image transformations are unnatural, and the human eye can clearly see the components of "fraud". In addition, *DeepHunter* uses pixel level constraints when keep valid examples, which are the low-level features of the image. When testing the model with deeper layers, the test effect is not good. In contrast, in *CAGFuzz*, *AEG* generates adversarial examples by adding small perturbations to the original examples that are not visible to the human eye, and the adversarial examples generated by *AEG* are natural and more confusing to the model. At the same time, the deep features are used in *CAGFuzz* to constrain the adversarial examples, and consequently the adversarial examples have effective test effect on the model with deep layers.

DeepTest [17] performs a tool for automated testing of DNN-driven autonomous cars. *DeepTest* does not consider the small perturbations on input examples and however maximizes the neuron coverage of a DNN using synthetic test images generated by applying different real transformations to a set of seed images. The image transformation method of *DeepTest* is the affine transformation (image translation, image scaling, image shearing and image rotation) in

DeepHunter. Therefore, *DeepTest* has the similar problem as *DeepHunter*.

In addition, many testing approaches for traditional software have also been adopted and applied to testing DL systems, such as MC/DC coverage [15], concolic test [54], combinatorial test [55] and mutation test [56]. Furthermore, various forms of neuron coverage [16] have been defined, and are demonstrated as important metrics to guide test generation. In general, these approaches do not consider adversarial examples and test DL systems from other aspects.

6 Conclusions and Suggestions for Future Work

We design and implement *CAGFuzz*, a coverage-guided adversarial generative fuzzing testing approach. *CAGFuzz* trains an adversarial example generator for a specified dataset. It generates adversarial examples for target DNN by iteratively taking original examples, generating adversarial examples and feedback of coverage rate, and finds potential defects in the development and deployment phase of DNN. We have done a lot of experiments to prove the effectiveness of *CAGFuzz* in promoting DNN coverage, discovering potential errors in DNN and improving model accuracy. The goal of *CAGFuzz* is to maximize the neuron coverage and the number of potential erroneous behaviors. The experimental results show that *CAGFuzz* can detect thousands of erroneous behaviors in advanced DNN models, which are trained on publicly popular datasets.

Several directions for future work are possible.

- At present, we only use neuron coverage to guide the generation of adversarial examples. Neuron coverage may not cover all the logic of DNN effectively. In the future, we can use multidimensional coverage feedback to improve the information that adversarial examples can cover.
- *CAGFuzz* adds perturbation information to the original example by mapping between two data domains. These perturbations are uncontrollable. In the future, the perturbation information can be added to the original example by feature control.
- This paper mainly studies image examples, and how to train effective adversarial example generator for other input forms, such as text information and voice information, is also a meaningful direction.

7 Acknowledgements

The work is supported by the National Natural Science Foundation of China under Grant No. 61572171, the Natural Science Foundation of Jiangsu Province under Grant No. BK20191297, and the Fundamental Research Funds for the Central Universities under Grant No. 2019B15414.

References

[1] X. Lei, G.-H. Tu, A. X. Liu, C.-Y. Li, and T. Xie, "The insecurity of home digital voice assistants-amazon alexa as a case study," *arXiv preprint arXiv:1712.03327*, 2017.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[4] P. Sermanet and Y. LeCun, "Traffic sign recognition with multi-scale convolutional networks.," in *IJCNN*, pp. 2809–2813, 2011.

[5] F. Zhang, J. Leitner, M. Milford, B. Upcroft, and P. Corke, "Towards vision-based deep reinforcement learning for robotic motion control," *arXiv preprint arXiv:1511.03791*, 2015.

[6] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[7] M. Latah and L. Toker, "Artificial intelligence enabled software-defined networking: a comprehensive overview," *IET Networks*, vol. 8, no. 2, pp. 79–99, 2018.

[8] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust physical-world attacks on deep learning visual classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1625–1634, 2018.

[9] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in *2007 Future of Software Engineering*, pp. 85–103, IEEE Computer Society, 2007.

[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*, pp. 1–18, ACM, 2017.

[11] X. Xie, L. Ma, F. Juefei-Xu, H. Chen, M. Xue, B. Li, Y. Liu, J. Zhao, J. Yin, and S. See, "Deephunter: Hunting deep neural network defects via coverage-guided fuzzing," *arXiv preprint arXiv: 1809.01266*, 2018.

[12] L. Fei-Fei, "Imagenet: crowdsourcing, benchmarking & other cool things," in *CMU VASC Seminar*, vol. 16, pp. 18–25, 2010.

[13] R. Merkel, "Software reliability growth models predict autonomous vehicle disengagement events," *arXiv preprint arXiv:1812.08901*, 2018.

[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*, pp. 739–743, ACM, 2018.

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

[16] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu, *et al.*, "Deepgauge: Multi-granularity testing criteria for deep learning systems," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pp. 120–131, ACM, 2018.

[17] 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*, pp. 303–314, ACM, 2018.

[18] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[19] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 427–436, 2015.

[20] E. D. Cubuk, B. Zoph, S. S. Schoenholz, and Q. V. Le, "Intriguing properties of adversarial examples," *arXiv preprint arXiv:1711.02846*, 2017.

[21] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, pp. 2223–2232, 2017.

[22] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[23] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[24] H. Li, H. Liu, X. Ji, G. Li, and L. Shi, "Cifar10-dvs: An event-stream dataset for object classification," *Frontiers in neuroscience*, vol. 11, p. 309, 2017.

[25] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.
[26] M. Zalewski, "American fuzzy lop," *URL: http://lcamtuf. coredump. cx/afl*, 2017.
[27] K. Serebryany, "libfuzzer a library for coverage-guided fuzz testing," *LLVM project*, 2015.
[28] S. Rawat, V. Jain, A. Kumar, L. Cojocar, C. Giuffrida, and H. Bos, "Vuzzer: Application-aware evolutionary fuzzing.," in *NDSS*, vol. 17, pp. 1–14, 2017.
[29] V. T. Pham, M. Bhme, A. E. Santosa, A. R. Cciulescu, and A. Roychoudhury, "Smart greybox fuzzing," *IEEE Transactions on Software Engineering, 2018, DOI:10.1109/TSE.2019.2941681*.
[30] I. 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*, pp. 2672–2680, 2014.
[31] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. Paul Smolley, "Least squares generative adversarial networks," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2794–2802, 2017.
[32] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition," in *International conference on machine learning*, pp. 647–655, 2014.
[33] A. Babenko, A. Slesarev, A. Chigorin, and V. Lempitsky, "Neural codes for image retrieval," in *European conference on computer vision*, pp. 584–599, Springer, 2014.
[34] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *arXiv preprint arXiv:1906.10742*, 2019.
[35] N. Ketkar, "Introduction to keras," in *Deep learning with Python*, pp. 97–111, Springer, 2017.
[36] Z. Zhao, D. Dua, and S. Singh, "Generating natural adversarial examples," *arXiv preprint arXiv:1710.11342*, 2017.
[37] P. Samangouei, M. Kabkab, and R. Chellappa, "Defense-gan: Protecting classifiers against adversarial attacks using generative models," *arXiv preprint arXiv:1805.06605*, 2018.
[38] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE transactions on neural networks and learning systems*, 2019.
[39] 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)*, pp. 372–387, IEEE, 2016.
[40] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," *arXiv preprint arXiv:1607.02533*, 2016.
[41] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 39–57, IEEE, 2017.
[42] S. Gan, C. Zhang, X. Qin, X. Tu, K. Li, Z. Pei, and Z. Chen, "Collafl: Path sensitive fuzzing," in *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 679–696, IEEE, 2018.
[43] J. Wang, B. Chen, L. Wei, and Y. Liu, "Superion: grammar-aware greybox fuzzing," in *Proceedings of the 41st International Conference on Software Engineering*, pp. 724–735, IEEE Press, 2019.
[44] C. Holler, K. Herzig, and A. Zeller, "Fuzzing with code fragments," in *Presented as part of the 21st {USENIX} Security Symposium ({USENIX} Security 12)*, pp. 445–458, 2012.
[45] R. Padhye, C. Lemieux, K. Sen, M. Papadakis, and Y. L. Traon, "Zest: Validity fuzzing and parametric generators for effective random testing," *arXiv preprint arXiv:1812.00078*, 2018.
[46] K. Serebryany, V. Buka, and M. Morehouse, "Structure-aware fuzzing for clang and llvm with libprotobuf-mutator," 2017.
[47] T. Petsios, A. Tang, S. Stolfo, A. D. Keromytis, and S. Jana, "Nezha: Efficient domain-independent differential testing," in *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 615–632, IEEE, 2017.
[48] A. Odena and I. Goodfellow, "Tensorfuzz: Debugging neural networks with coverage-guided fuzzing," *arXiv preprint arXiv:1807.10875*, 2018.
[49] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.
[50] D. L. Rosenband, "Inside waymo's self-driving car: My favorite transistors," in *2017 Symposium on VLSI Circuits*, pp. C20–C22, IEEE, 2017.
[51] I. Goodfellow and N. Papernot, "The challenge of verification and testing of machine learning," *Cleverhans-blog*, 2017.
[52] M. Böhme, V.-T. Pham, and A. Roychoudhury, "Coverage-based greybox fuzzing as markov chain," *IEEE Transactions on Software Engineering*, vol. 45, no. 5, pp. 489–506, 2017.
[53] J. Wang, B. Chen, L. Wei, and Y. Liu, "Skyfire: Data-driven seed generation for fuzzing," in *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 579–594, IEEE, 2017.
[54] Y. Sun, M. Wu, W. Ruan, X. Huang, M. Kwiatkowska, and D. Kroening, "Concolic testing for deep neural networks," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pp. 109–119, ACM, 2018.
[55] L. Ma, F. Zhang, M. Xue, B. Li, Y. Liu, J. Zhao, and Y. Wang, "Combinatorial testing for deep learning systems," *arXiv preprint arXiv:1806.07723*, 2018.
[56] L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. Juefei-Xu, C. Xie, L. Li, Y. Liu, J. Zhao, *et al.*, "Deepmutation: Mutation testing of deep learning systems," in *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 100–111, IEEE, 2018.

Pengcheng Zhang received the Ph.D. degree in computer science from Southeast University in 2010. He is currently an associate professor in College of Computer and Information, Hohai University, Nanjing, China, and was a visiting scholar at San Jose State University, USA. His research interests include software engineering, service computing and data mining. He has published in premiere or famous computer science journals, such as IEEE TBD, IEEE TETC, IEEE TSC, IST, JSS, and SPE. He was the co-chair of IEEE AI Testing 2019 conference. He served as technical program committee member on various international conferences. He is a memeber of the IEEE.

Qiyin Dai received the bachelor's degree in computer science and technology from nanjing university of finance and economics in 2018. He is currently working toward the M.S. degree with the College of Computer and Information, Hohai University, Nanjing, China. His current research interests include data mining and software engineering.

Patrizio Pelliccione is Associate Professor at the Department of Information Engineering, Computer Science and Mathematics - University of L'Aquila (Italy), and he is also Associate Professor at the Department of Computer Science and Engineering at Chalmers University of Gothenburg (Sweden). He got his PhD in 2005 at the University of L'Aquila (Italy) and from February 1, 2014 he is Docent in Software Engineering, title given by the University of Gothenburg (Sweden). His research topics are mainly in software engineering, software architectures modelling and verification, autonomous systems, and formal methods. He has co-authored more than 130 publications in journals and international conferences and workshops in these topics. He has been on the program committees for several top conferences, he is a reviewer for top journals in the software engineering domain, and he organized as program chair international conferences. He is very active in European and National projects. More information is available at http://www.patriziopelliccione.com.