CAGFuzz: Coverage-Guided Adversarial Generative Fuzzing Testing for Image-based Deep Learning Systems

Pengcheng Zhang, Bin Ren, Qiyin Dai, and Hai Dong

Abstract—Deep Neural Network (DNN) driven technologies have been extensively employed in various aspects of our life. Nevertheless, the applied DNN always fails to detect erroneous behaviors, which may lead to serious problems. Several approaches have been proposed to enhance adversarial examples for automatically testing deep learning (DL) systems, such as image-based DL systems. However, the approaches contain the following two limitations. First, existing approaches only take into account small perturbations on adversarial examples, they design and generate adversarial examples for a certain particular DNN model. This might hamper the transferability of the examples for other DNN models. Second, they only use shallow features (e.g., pixel-level features) to judge the differences between the generated adversarial examples and the original examples. The deep features, which contain high-level semantic information, such as image object categories and scene semantics, are completely neglected. To address these two problems, we propose *CAGFuzz*, a Coverage-guided Adversarial Generative Fuzzing testing approach for image-based DL systems. *CAGFuzz* is able to generate adversarial examples for mainstream DNN models to discover their potential errors. First, we train an Adversarial Example Generator (*AEG*) based on general datasets. *AEG* only considers the data characteristics to alleviate the transferability problem. Second, we extract the deep features of the original and adversarial examples, and constrain the adversarial examples by cosine similarity to ensure that the deep features of the adversarial examples remain unchanged. Finally, we use the adversarial examples to retrain the models. Based on several standard datasets, we design a set of dedicated experiments to evaluate *CAGFuzz*. The experimental results show that *CAGFuzz* can detect more hidden errors, enhance the accuracy of the target DNN models, and generate adversarial examples with higher transferability.

Index Terms—Deep neural network; Fuzz testing; Adversarial example; Coverage criteria.

1 Introduction

AI technologies have become more prominent in our lives. In many applications, we can observe the traces of deep neural networks (DNN), such as automatic driving [1], [2], intelligent robotics [3], smart city applications [4] and AI-enabled Enterprise Information Systems [5]. In this paper, we term this type of applications as deep learning (DL) systems. DL systems can be generally categorized into image-based, text-based, audio-based and video-based systems, according to their served objects. In this research, we only focus on image-based DL systems.

Many different types of DNN are embedded in mission and safety-critical applications, such as automatic driving [1] and intelligent robotics [3]. This brings new challenges, since predictability, correctness, and safety are especially crucial for these types of DL systems. These safety and mission-critical applications deploying DNN without comprehensive testing could cause serious problems. For instance, in automatic driving systems, if the deployed DNN have not recognized the obstacles ahead timely and correctly, it might lead to serious consequences, such as vehicle damage and even human death [6].

- *P. Zhang, B. Ren, and Q. Dai are with College of Computer and Information, Hohai University, Nanjing, China. E-mail: pchzhang@hhu.edu.cn;*
- *H. Dong is with School of Computing Technologies, RMIT University, Melbourne, Australia E-mail: hai.dong@rmit.edu.au*

Manuscript received XXX, XXXX; revised XXX, XXXX.

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. In contrast, DL systems automatically learn their models and corresponding parameters from data. For traditional software systems, code or control-flow coverage is utilized to guide the testing process [7]. In comparison, the logic of the DL systems is not encoded by control flows and thus 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 interventions. Consequently, most traditional software testing methodologies are not suitable for testing DL systems. As highlighted in [8], [9], research on developing new testing techniques for DL systems is urgently needed.

Obviously, it is unthinkable to exhaustively test every feasible input of the DL systems. Recently, an increasing number of researchers have contributed to testing DL systems with a variety of approaches [8], [9], [10], [11], [12], [13]. The main idea of these approaches is to enhance input examples of test datasets by diverse techniques. Some approaches, e.g., *DeepXplore* [8], are based on multiple DNN models, where decision boundaries are predefined among the models. Adversarial examples are generated by changing the gradient direction in a decided model. Some approaches, e.g., *DeepHunter* [9], employ a metamorphic

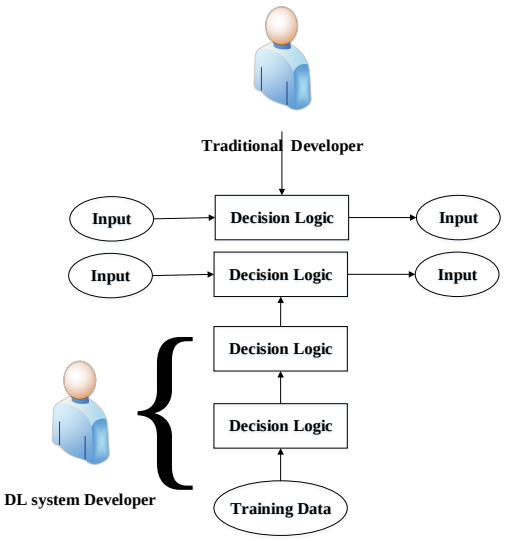


Fig. 1: Comparison between traditional and DL system development

mutation strategy to generate new adversarial examples. Other approaches, e.g., *DeepGauge* [12], propose new coverage criteria for DNN. These coverage criteria can be used as guidance for generating adversarial examples. While these approaches make some progresses on testing DL systems, they still suffer the following two main problems:

1) *Unsatisfied transferability of the generated adversarial examples.* Most approaches [9], [14] neglect the influence of small perturbations when generating adversarial examples. Several approaches such as [8], [15] consider small perturbations. However, the transferability [16] of the examples generated by these approaches appears to be unsatisfied. Through our experiments, we found that, when the adversarial examples generated for a specific model are used to retrain other similar models, the increased accuracy for the other models cannot maintain the same level as it for the specific model. For example, the accuracy of the VGG-16 model can be improved by 28.2% by using the adversarial examples generated for the VGG-16 model. In contrast, the enhanced accuracy of the VGG-16 model is only 21.8% (i.e. dropped by 22.69%) if it is trained upon the adversarial examples generated for the VGG-19 model. The common solution is to generate adversarial examples for each model to more effectively improve their accuracy. However, our experiment finds that the generation of 50000 adversarial examples (based on the CIFAR-10 dataset) takes nearly 3 hours (approx. 0.25s per example) each time. In addition, each set of newly generated adversarial examples need 113M of storage space. Therefore, adversarial example generation for each single model might not be a reasonable choice when multiple models need to be trained and time/storage resources are limited.
2) *Low accuracy caused by shallow features.* State-of-the-art adversarial example generation approaches use shallow features, such as pixel-level features, to judge the differences between the adversarial examples and the original examples. The deep features containing high-level semantic information, such as image object category and scene semantics, are completely neglected. For instance, in their study, Xie et al. [9] use L_0 and L_∞ to limit the pixel-level changes of the adversarial examples. However, such shallow features can only represent the visual consistency between the adversarial examples and the original examples, and cannot guarantee the deep features consistency between the adversarial examples and the original examples. This might lead to low accuracy when testing the networks with deep layers.

To address the problems aforementioned, we propose a new fuzzing testing approach for image-based DL systems, called *CAGFuzz* (Coverage-guided Adversarial Generative Fuzzing)[1]. The goal of *CAGFuzz* is to explore the use of neuron coverage to generate adversarial examples with minimal perturbations for the target DNN. Meanwhile, we aim to generate examples with higher transferability. In other words, the examples can be used to effectively test multiple DNN models. In summary, the major 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 datasets.* The goal of *CycleGAN* [17] is to transform *image A* to *image B* with different styles. Based on *CycleGAN*, we aim to transform *image B* back to *image A*, and obtain *image A'* similar to the original *image A*. Consequently, we combine the two generators with the opposite functions of *CycleGAN* as our adversarial example generator. The adversarial examples generated by *AEG* can add small perturbations to the original examples. *AEG* is trained based on general datasets and does not rely on any specific DNN model, which has higher transferability than state-of-the-art approaches. The experimental results demonstrate the enhanced transferability of the *AEG* generated adversarial examples. They not only improve the accuracy of multiple models effectively, but also make the accuracy of the target model 3.39% higher than that of a typical model-based approach, i.e., *DeepXplore*.
- *We extract the deep features of the original examples and the adversarial examples, and make them as similar as possible by similarity measurement.* We use VGG-19 network [18] to extract the deep features of the original examples and the adversarial examples, and employ cosine similarity measurement to ensure that the deep features of the adversarial examples are consistent with the original examples as much as possible. At the same time, the deep features can facilitate the adversarial examples generated by *CAGFuzz* to obtain better results compared with other approaches when testing the neural networks with deeper layers.

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

- *We design a series of experiments to evaluate the CAGFuzz approach based on several public datasets.* The experiments validate that *CAGFuzz* can effectively improve the transferability of the generated adversarial examples. Meanwhile, it is proved that the adversarial examples generated by *CAGFuzz* can detect hidden errors in the target DNN model. Furthermore, the accuracy of the DNN models retrained by *AEG* have been significantly improved.

The rest of the paper is organized as follows. Section 2 provides some basic concepts including coverage-guided grey-box fuzzing, AEG, VGG-19 and neuron coverage. The coverage-guided adversarial generative fuzzy testing approach is provided in Section 3. In Section 4, we use three popular datasets (MNIST [19], CIFAR-10 [20], and ImageNet [21]) to validate our approach. Existing works are discussed in Section 5. Finally, Section 6 concludes the paper.

2 Preliminaries

The principles of *Coverage-guided Grey-box Fuzzing*, *Adversarial Example Generator*, *VGG-19 Network Structure* and *Neuron Coverage* are introduced in Section 2.1, Section 2.2, Section 2.3, and Section 2.4 respectively. Finally, other important terminology definitions used in the paper are described in Section 2.5.

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 [22]. Many state-of-the-art CGF approaches, such as AFL [23], libFuzzer [24] and VUzzer [25], have been widely used and proved to be effective. The state-of-the-art CGF approaches mainly consist of the following three parts:

- *Mutation:* According to the difference of the target application program and data format, the corresponding test data generation method is chosen. 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.

Due to the difference between DL systems and traditional software systems, traditional CGF cannot be directly applied to DL systems. In our approach, CGF is improved to be suitable for DL systems. The state-of-the-art CGF approaches mainly consist 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, since the test is basically fed with the same format of images, the adversarial examples with higher coverage are selected and added into the processing pool to maximize the neuron coverage of the target DL systems. This paper aims to provide a quality assurance tool for DNN development and deployment by designing an effective CGF framework.

2.2 Adversarial Example Generator

Adversarial Example Generator (*AEG*) is an important part of our approach. To improve the stability and security of target DL systems, *AEG* provides effective adversarial examples to detect potential errors. 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 example generation using GAN [26]. 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 for various target DL systems, e.g., some DL systems relying on labelled data while others not, we choose *CycleGAN* [17] as the training model of *AEG*, since *CycleGAN* does not require the paired examples and label information. Two sets of generators and discriminators of *CycleGAN* are used to train alternately, and the two generators are combined to form an *AEG*. During *AEG* training, the adversarial loss and the cycle consistency loss are the important guarantee to keep the image similarity between adversarial examples and the seed examples. Fig. 2 shows the schematic diagram of AEG. The part in the red box shows the components of AEG.

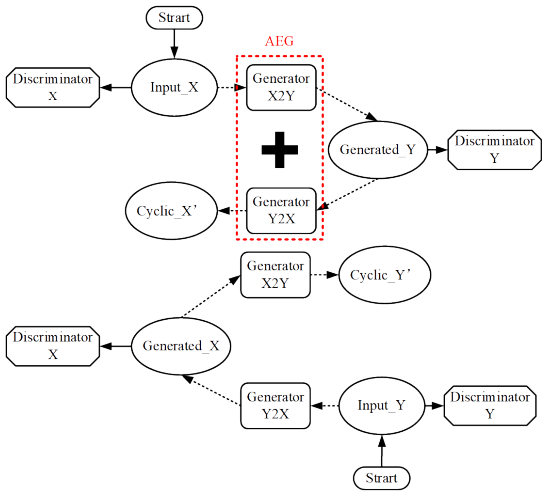


Fig. 2: Components of adversarial example generator

2.3 VGG-19 Network Structure

As a feed-forward neural network, the last layer of CNN has m neurons. Each neuron generates a scalar. The output of M neurons can be regarded as a vector v. Now all of them are connected to one neuron. The output of this

neuron is $wv + b$, which is a continuous value and can deal with regression prediction problems. The function of CNN is similar to that of BP neural network. Instead of explicit mathematical expressions, the mapping relationship between input and output is implied in sample data. Convolutional Neural Network (CNN) has remarkable capability to extract deep feature and express semantics. Those deep features and expressions are more effective than traditional image features, therefore, the CNN model trained by large image databases has better generalization capability [27]. The CNN models can extract deep features from the images for semantic segmentation, target detection and image retrieval. The structure of VGG-19 [18] convolution network is shown in Fig. 3. There are 19 layers in VGG-19, including 16 convolution layers, i.e., 2 Convl1-Convl2 and 4 Convl3-Convl5, and 3 fully connected layers, including Fc6, Fc7, and Fc8. The works in [28], [29] show that the VGG-19 network can extract deep features from images, and it can be used to identify the similarity between images. In this paper, the output of the last fully connected layer is fused into a 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.

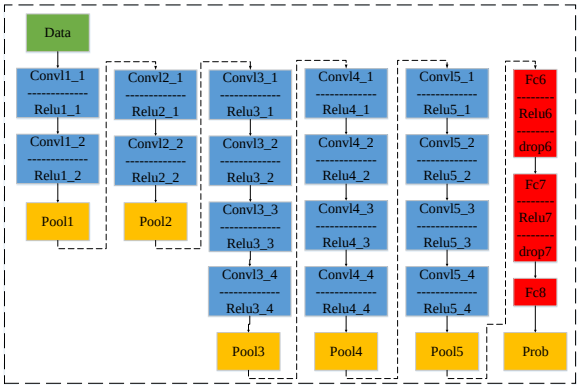


Fig. 3: Structure of VGG-19 network for extracting deep features of target images

2.4 Neuron Coverage

Pei et al. [8] initially propose neuron coverage as a measure for testing DL. They define neuron coverage of a set of test inputs as the ratio of the number of uniquely activated neurons for all the test inputs to the total number of neurons in the DNN.

Let $N = \{n_1, n_2, ..., n_p\}$ be all the neurons in a DNN, where p is the number 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 the 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:

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

2.5 Other Terminology Definitions

The other important terminology definitions used in this paper are described below.

Transferability. Szegedy et al. [16] bring up the concept of transferability, indicating the capability of adversarial examples generated for one model to stay adversarial for other models. In this paper, transferability refers to whether the same set of adversarial examples are effective for enhancing the accuracy of multiple DNN models to perform the same task by retraining the models with the adversarial examples.

Deep Features. Hou et al. [30] name the output of the intermediate layer of a DNN as the deep features of an image, given the image as the input of the DNN. In this paper, deep features of an image are the output of a fully connected layer of a CNN model given the image vector as the input. In our approach, the output of Fc7 layer of the VGG-19 network is used as the deep features, represented as a 4096 dimensional vector.

Errors. In the field of software engineering, Salfner et al. [31] define an error as a situation when a system's state deviates from the correct state. In this paper, an error is defined as the inconsistency between the original input and the output of a DNN model caused by training the model with the adversarial examples of the input. For example, given an original example x and a corresponding adversarial examples x_{adv} generated by an adversarial attack, the error refers to $DNN(x) \neq DNN(x_{adv})$, where $DNN(\cdot)$ represents the label of the output of the DNN model.

3 The *CAGFuzz* Approach

In this section, we first give an overview of our approach (Section 3.1). Second, the pre-treatment of our approach is described in Section 3.2, including data collection and *AEG* training. Section 3.3 describes the main 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 components of DL systems are DNNs with various structures and parameters. In this section, we will discuss how to test a DNN. Here we focus on DNNs whose input is images. 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. 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 steps described as follows:

- The *first step* is the data collection and *AEG* training. For each dataset, the data is divided into two subsets with balanced example numbers and then employed as the input of *CycleGAN* to train *AEG*. These examples are transferred to the processing pool after configuring their priority according to their time to join the processing pool. We make use of this processing pool as the initial input for fuzzy testing.

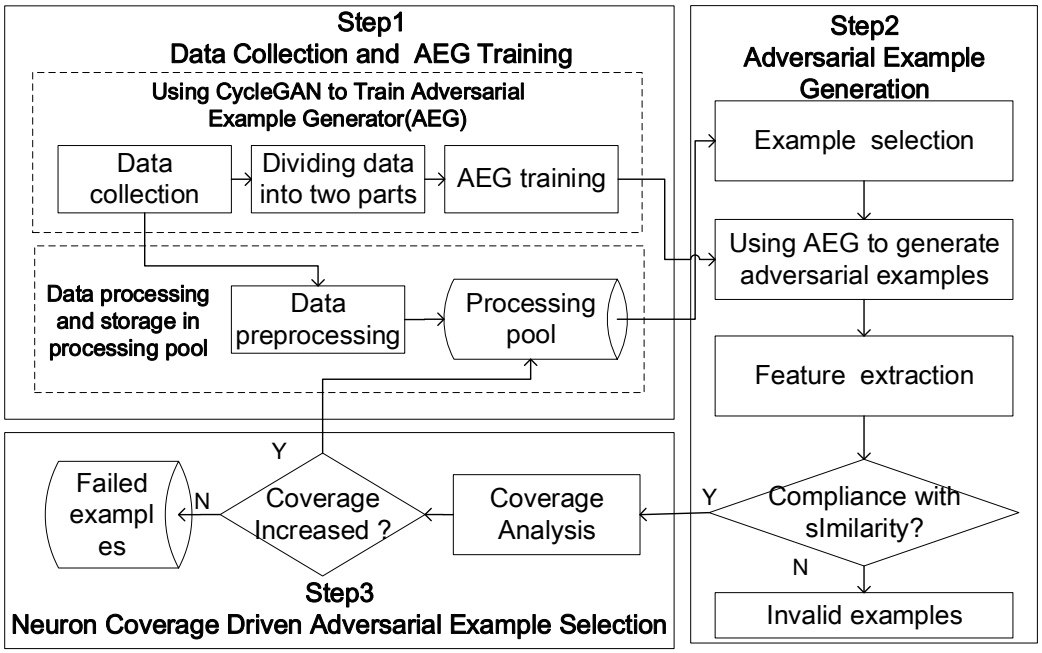


Fig. 4: Coverage-Guided Adversarial Generative Fuzzing testing approach

- The *second step* is the adversarial example generation. Each time a prioritized raw example is selected from the processing pool and utilized as the input of *AEG* to generate adversarial examples. Deep features are then employed to determine which adversarial examples should be saved. First, we adopt 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 features and can be saved.
- The *third step* is to adopt neuron coverage to guide the generation process. Each adversarial example generated in the second step is provisioned as an input to the DNN under test for coverage analysis. If new coverage occurs, the adversarial example will be placed into the processing pool as a 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 target 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 adversarial examples that improve the coverage of the target DNN.

We need to process the dataset before the fuzzing process. The dataset is divided into two equal subsets (Line 1) to train AEG (Line 2). All the examples in the dataset are pre-processed (Line 3) and stored in the processing pool (Line 4). During each iteration (Line 5), the original example $parent$ is selected from the processing pool according to the selection priority (Lines 6-7). Then, multiple adversarial examples are generated for each original example $parent$ (Line 8). For each generation, 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 the original example are sorted from high to low in similarity, and the top-k of them are selected as the target examples (Line 13). The neuron coverage of the top-k adversarial examples is calculated and analyzed to determine whether the adversarial examples should be saved (Line 15). If the adversarial examples increase the neuron coverage of the target DNN, they will be stored in the processing pool with a specified selection priority (Lines 16-19). The selection priority solution is detailed in Section 3.3.1.

3.2 Data Collection and AEG Training

3.2.1 Data Collection

We define the target task of *CAGFuzz* as an image classification problem. Image classification is the core mission of many existing DL systems. The first step of *CAGFuzz* is to choose an image classification oriented 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

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

Input: D: Corresponding datasets,
DNN: Target Deep Neural Network,
N: The maximum number of iterations,
$N1$: The number of new examples generated,
K: Top-k parameter
Output: Test example set for increasing coverage

```
X, Y = Divide(D);
Train AEG through X and Y
T = Preprocessing(D);
T serves as the initial processing pool
while number of iterations < N do
    S = HeuristicSelect(T);
    parent = Sample(S);
    while number of generations < N1 do
        data = AEG(parent);
        Fp, Fd = FeatureExtraction(parent,data);
        Similarity = CosineSimilarity(Fp,Fd);
    end while
    Select top-k examples from all new examples;
    while number of calculations < K do
        cov = DNNFeed(data);
        if IsNewCoverage(cov) then
            Add data to the processing pool
            Set selection priority for data;
        end if
    end while
end while
Output all the examples in the processing pool as a test example set;
```

examples. During the process of fuzzing, the fuzzer selects an original example from the processing pool according to the priority to perform the fuzzing operation. Second, the dataset is divided into two sets of uniform domains. According to the domains, it is used as the input of *CycleGAN* to train *AEG*.

3.2.2 Training Adversarial Example Generator

Traditional fuzzers mutate the original examples by flipping bits/bytes, cross-inputting files, and swapping blocks to achieve the effect of fuzziness. However, mutation of DNN inputs using these methods is unachievable or invalid, and might produce a large number of invalid testing examples. At the same time, how to grasp the degree of mutation is also a question worth further study. If the mutation makes limited changes, the newly generated examples might be almost unchanged. Although these changes may be meaningful, the possibility of employing the new examples for DNN error detection is very low. On the other hand, if the mutation makes dramatic changes, more DNN errors may be found. However, these changes might be unrealistic in the real application environment. That is, the newly generated examples are also invalid.

We propose a new strategy that employs *AEG* for mutations. Given an image example x, *AEG* generates an adversarial example x', and the deep features of x' are consistent with those of x, where the adversarial perturbations that are not human-observable are added. There are two reasons

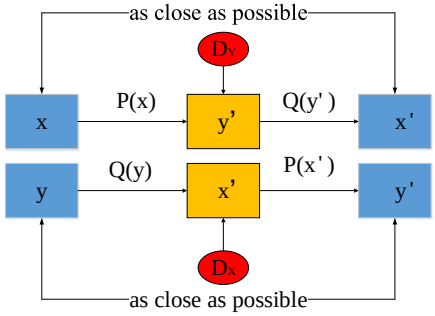


Fig. 5: Relationship between two mapping functions in training *AEG*

why we choose *CycleGAN* as the training model of *AEG*: GAN technology is one of the most effective and convenient image generation methods; *CycleGAN* does not rely on any additional information to generate adversarial examples.

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

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

$$\min_P \max_D YV(P, D_Y, X, Y) = E_{y \sim P_{data}(y)}[logD_Y(y)] + E_{x \sim P_{data}(x)}[log(1 - D_Y(P(x)))] \quad (2)$$

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 the characteristics of data domain Y to the original example x of data domain X. Simultaneously, there is an adversarial discriminator D_Y to distinguish the real example y in data domain Y and the generated adversarial example y'. The goal of the objective function is to minimize mapping function P and maximize adversarial discriminator D_Y. Similarly, for map-

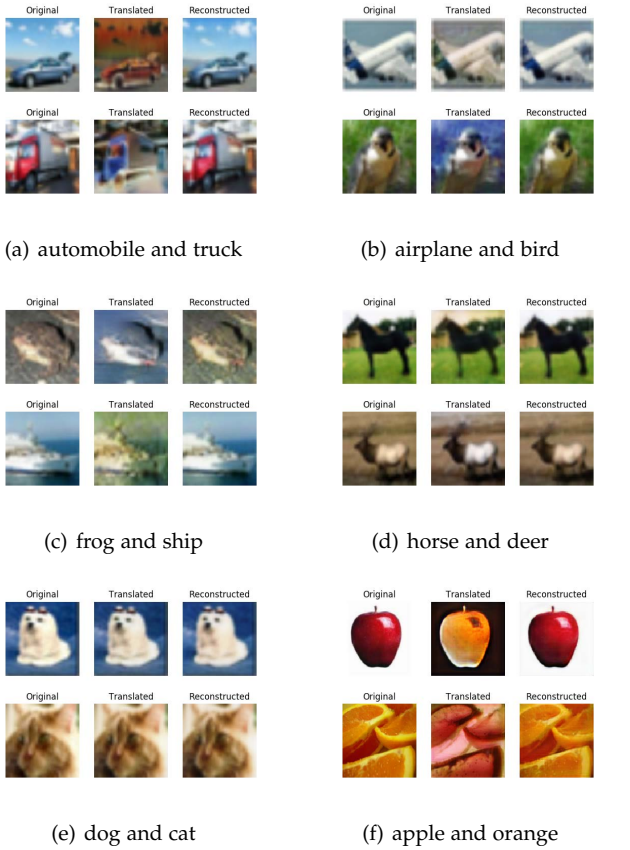
ping function Q and the target function set by adversarial discriminator D_X, the objective function is defined as:

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

Cycle Consistency Loss. We add perturbations to the original example by using the aforementioned adversarial loss function. However, the degree of mutation of these perturbations is large, which is prone to generate invalid adversarial examples. To avoid this problem, we add constraints to the perturbations, and control the degree of mutations through the cycle consistency loss. In this way, the perturbations added to the original example are invisible. For instance, an adversarial example y' is generated from an example x of data domain X by the mapping function P, and then y' is passed to the mapping function Q to generate its 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:

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

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 the two generators with the opposite functions into our *AEG*. The sample effect of *AEG* is shown in Fig. 6. The images on the leftmost column are the original examples, on the middle column are the transformed examples of the original examples, and on the rightmost column are the reconstructed examples. We choose the reconstructed examples as the adversarial examples. First, larger perturbations are added to the original example. Second, the degree of mutation is controlled by the reverse reconstruction to generate the adversarial examples with smaller perturbations.



(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.

3.3 Adversarial Example Generation

3.3.1 Example Priority

The priority of the example determines which examples should be first selected next time. We adopt a probabilistic 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 is added into the processing pool.

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

3.3.2 Deep Feature Extraction

To ensure the generated adversarial examples as meaningful as possible, we extract the deep features of the original examples and adversarial examples and control their differences within a certain range. The deep feature recognition ability and semantics expression ability of CNN are more remarkable. Hence, we select the VGG-19 network to extract the deep features from examples. The deep features in the VGG-19 model are extracted in hierarchies. 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 can better represent images 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 fully connected layer (Fc8 layer in Fig. 3) as the targeted deep features, and the dimension of the deep features 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 the adversarial examples is $T = \{a_1, a_2, ..., a_n\}$, which inherits the deep feature vector of the original example by the feature extraction method mentioned above. The dimension of each feature vector is 4096. Let us assume 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 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 the adversarial example and the original example. The formula is described as follows:

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

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

To control the size and improve the mutation quality of adversarial examples, we select the top-k adversarial examples whose cosine similarity is sorted descendingly 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 analysis.

3.4 Neuron Coverage Driven Adversarial Example Selection

Without using neuron coverage as a guiding condition, the adversarial examples generated by *AEG* are not purposeful, and it is impossible to learn about whether the adversarial examples are effective or not. If the generated adversarial examples cannot generate new coverage of a DNN to be tested, these adversarial examples can only simply expand the dataset, rather than effectively detecting the potential errors of the DNN. To make matters worse, mutations in these adversarial examples may bury other meaningful examples in a fuzzy queue, and thus significantly reduce 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 mutations.

After each round of generation and similarity screening, all valid adversarial examples are used as the input of the DNN to be tested for neuron coverage analysis. If the adversarial examples generate higher neuron coverage, we will set the priority for the adversarial examples and store them in the processing pool for further mutations. For instance, 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 determine that the adversarial example generates new coverage.

4 Experimental Evaluation

We make use of 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 to explore the following three main research questions:

- *RQ1*: Do the adversarial examples generated by *CAGFuzz* have higher transferability than the adversarial examples generated by the existing models?
- *RQ2*: Can *CAGFuzz* find potential errors in the target network?
- *RQ3*: Can *CAGFuzz* improve the accuracy of the target network by adding adversarial examples into the training set?

4.1 Experimental Design

4.1.1 Experimental Environment

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 framework	Keras

4.1.2 Datasets and Corresponding DNN Models

With the evaluation purpose, we adopt three popular and commonly used datasets containing different types of data: MNIST [19], CIFAR-10 [20], and ImageNet [21]. 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. Among them, the models for CIFAR-10 are trained by ourselves, and the models for ImageNet employ the default weights and network structure provided by the Keras framework [2], so the models with the same names in the table are actually two different models.

MNIST [19] 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 adversarial examples. We aim to imitate the research of Lecun et al. **[ref]** and construct three different kinds of neural networks based on LeNet family, namely LeNet-1, LeNet-4, and LeNet-5.

CIFAR-10 [20] is a set of images used to train classification models. It contains $32*32*3$ pixel three-channel images, including ten different kinds of objects (such as aircraft, cats, trucks, etc.). The dataset contains 50,000 training examples and 10,000 adversarial examples. Due to the large amount of data and high complexity of CIFAR-10, its classification task is more challenging 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 [21] 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 1 million images) and test data

2. https://keras.io/

(about 50 thousand images). Therefore, for any automated testing tool, working on ImageNet-sized datasets and DNN models is a demanding task. Because of 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: Datasets and DNN models

Dataset	Description	Model	#Layer	#Neuron	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	12426	93.39
		VGG-19	19	13706	93.20
		ResNet-20	70	4861	94.53
ImageNet	1000-class large scale datasets	VGG-16	16	14888	90.1*
		VGG-19	19	16168	90.0*
		ResNet-50	176	94059	92.1*

* The top-5 test accuracy of pretrained DNN model in [32].

4.1.3 Baseline Approaches

As surveyed in [33], there are several open-source tools for testing machine learning applications, such as *Themis* [3], *mltest* [4], and *torchtes* [5]. None of them focus on generating adversarial examples. Thus, to measure the performance of *CAGFuzz*, we select the following approaches as our baseline approaches:

- *FGSM* (Fast Gradient Sign Method) [15]. There are many model-based adversarial example generation approaches, such as [16], [34], [35]. Most of them are variants of *FGSM* and their implementation principles are basically similar, whilst the frequencies of perturbation insertion are varied. After an in-depth analysis, we believe *FGSM* can be used to generate adversarial examples that meet the experimental requirements.
- *DeepHunter* [9] - an automated fuzz testing framework for hunting potential errors of general-purpose DNN. *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* [8] - 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* [9], we have implemented the eight image transformation methods mentioned in *DeepHunter*, and we use the eight methods to replace *DeepHunter* for later experimental evaluation. The source code of *FGSM* and *DeepXplore* can be found on GitHub. These tools are utilized for the forthcoming experimental evaluation.

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

4.2 Experimental Results

4.2.1 Transferability

To answer *RQ1*, we compare *CAGFuzz* with the existing model-based adversarial example generation approach *FGSM*. *FGSM* is a simple approach, in which no neuron coverage based technique is utilized. For the fairness of comparison, we enhance *FGSM* by facilitating it with the neuron coverage analysis. In this way, *FGSM* employs the same coverage-guided test approach as *CAGFuzz* does. For transferability assessment, the accuracy of the models varies more remarkably in small example data sets than it on large data sets, which is more conducive to analysis and comparison. Therefore, a few data examples are randomly selected from the original data set for evaluation. We choose MNIST and CIFAR-10 datasets 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 dataset have a large scale of weight parameters, 10 training examples are insufficient 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.

For the LeNet-1 model, we respectively employ *FGSM* and *AEG* to generate an adversarial example for each example in the training set. First, the original dataset is used to train and test the LeNet-1 model. We set the number of epochs 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.

Similar to generate adversarial examples for LeNet-1, we perform the same training process for LeNet-4 and LeNet-5. Since the model accuracy value varies after each iteration of training, we train each model 5 times in the same parameter setting and take the average of the accuracy values after each training as the final accuracy of our experiments. For instance, the accuracy values of the ResNet-20 model trained on the adversarial examples generated by FGSM-R20 are respectively 49%, 35%, 42%, 34% and 40% after each time of training. Its final accuracy value is 40%. Table 3 shows the accuracy of the three models on the original dataset, FGSM-Le1, FGSM-Le4, FGSM-Le5, and CAGFuzz-dataset. Among them, "FGSM-Le1" refers to the dataset generated by *FGSM*, and "CAGFuzz-dataset" refers to the dataset generated by *CAGFuzz*. From the table, it can be seen that the accuracy of each model based on *FGSM* generated adversarial examples for that model is improved more strikingly than its accuracy based on the examples generated for the other models. We also find that, after retraining three models based on CAGFuzz-dataset, the accuracy of the models are all remarkably increased and higher than the accuracy based on the *FGSM* datasets. By following the same way, results are obtained based on the CIFAR10 dataset, and the final results are shown in Table 4. We can see that, after retraining the three models based on CAGFuzz-dataset, the accuracy of

each model is mostly higher than the maximum accuracy of the retrained model based on *FGSM*. The ResNet-20 model is the only exception, whose accuracy based on *CAGFuzz* is 0.8% lower than FGSM-R20 but much higher than FGSM-V16 and FGSM-V19.

▷ *Answer to* ***RQ1****:* Based on the MNIST and CIFAR-10 datasets, we prove that the adversarial examples generated by *CAGFuzz* can remarkably enhance the accuracy of DNN models than *FGSM* in general. The latter can only achieve reasonable accuracy improvement when specially generating data for each model. In contrast, the former can dramatically improve the accuracy of all the models, so its transferability is higher.

TABLE 3: The accuracy of the three models on the MNIST dataset, the adversarial examples generated by *FGSM* and *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 dataset, the adversarial examples generated by *FGSM* and *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**

4.2.2 Erroneous Behavior Discovery

To answer *RQ2*, we sample the examples correctly classified by the DNN models from the test set of each dataset. Based on these correctly classified examples, we generate adversarial examples for each example through *AEG*. We can confirm that all the generated adversarial examples are classified correctly, because the deep features between the adversarial examples and the original examples are consistent. The "positive examples" generated by *AEG* are input into the corresponding DNN for classification. If there are classification errors, a potential defect of the classification model can be found. We define an original correct example as $Image_{orig}$ and its corresponding adversarial examples as $Image_{adv} = \{Image_1, Image_2, ..., Image_n\}$. Since the original example $Image_{orig}$ is classified correctly in the target DNN model, $Image_i$ should also be classified correctly, where $Image_i \in Image_{adv}$. If the classification of an adversarial example is incorrect, we consider it as an erroneous behavior of the target DNN.

We choose a quantitative measure to evaluate the effectiveness of *CAGFuzz* in detecting erroneous behaviors in different models. We choose 2000 examples, which are verified to be correct from each dataset. Then we use the four approaches to mutate these examples, and generate 2000 adversarial examples for our experiments. Table 5 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* [15], *DeepHunter* [9], and *DeepXplore* [8] in each dataset.

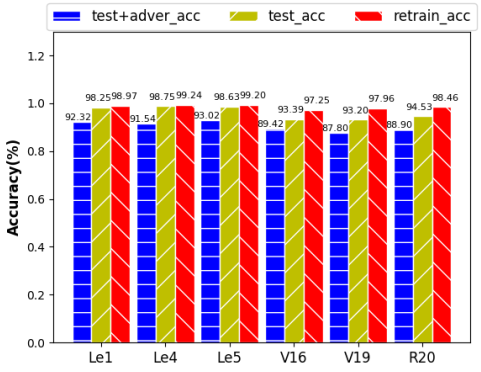


Fig. 7: Improvement of accuracy after model retraining.

TABLE 5: Number of erroneous behaviors reported by *FGSM* [15], *DeepHunter* [9], *DeepXplore* [8], and *CAGFuzz* across 2000 adversarial examples.

datasets	*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 5, *DeepXplore*'s performance in each dataset is the worst. Its total number of potential errors detected from the three datasets is only 72. Compared with the other three approaches, *CAGFuzz* demonstrates an outstanding ability to find potential errors in the models.

▷ *Answer to* ***RQ2****:* With neuron coverage guided adversarial examples, *CAGFuzz* can find more potential model errors.

4.2.3 Accuracy

To answer *RQ3*, we add adversarial examples generated by *CAGFuzz* into the training set to retrain the DNN models and measure whether it can improve their accuracy. We select the MNIST and CIAR-10 datasets as our experimental datasets. We employ LeNet-1, 4, 5, VGG-16, VGG-19, and ResNet-20 models as the experimental models. We retrain the DNN models by combining 65% of the generated adversarial example set and the original training set, and then validate the DNN models with the combination of the remaining 35% of the adversarial example set and the original validate set. Due to the space limitation, we abbreviate the model names. For instance, 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. 7, "test_acc" represents the accuracy of a model on the original test set, "test+adver_acc" represents the accuracy of a model on the new test set with the adversarial examples (the model is still the original one), and "retrain_acc" represents the accuracy of a model on the new test set after retraining the

model with the adversarial examples. It can be seen that, from the comparison of "test_acc" and "test+adver_acc", the robustness of the original models is low. After the adversarial examples being added into the test set, the accuracy of the models decreases evidently. For instance, the accuracy of the LeNet-5 model decreases from 98.63% to 93.02%. The comparison between "test_acc" and "retrain_acc" shows that the accuracy of the models has been greatly improved after retraining them with the adversarial examples. For instance, the accuracy of the VGG-19 network has been increased from 93.2% to 97.96%. In general, we can see that *CAGFuzz* can enhance the accuracy of the models, especially for the models with deeper layers.

We further analyze the accuracy of the retrained models and the original models during the training process, and evaluate the validity of the adversarial examples generated by *CAGFuzz* from the perspective of validation accuracy variation. Fig. 8 shows the changes of validation accuracy of those models during the training process. The original structure parameters and learning rate of each model are kept unchanged, and the aforementioned new training dataset 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 all the retrained models is higher than that of their corresponding original models. The convergence speed of the retrained models is also apparently faster. In addition, it can be found from the figure that the retrained models are more stable and show smaller variation ranges during the training process. In addition, we can observe that the trend of the retrained models are basically consistent with the original models, which implies that the accuracy of these models can be greatly improved without affecting their internal structure and logic. For instance, in Fig. 8(d), the accuracy of the original model drops suddenly when epoch = 6, and the retrained model follows the same pattern simultaneously. In Fig. 8(f), the accuracy of the original model and the retrained model both appears as a three-stage rising curve.

To further validate our approach, we pre-train these models on the MNIST and CIFAR-10 datasets. We further expand the training data by adding the same number of generated adversarial examples, and train the DNN in 5 epochs. The comparison results are shown in Fig. 9. It can be found that *CAGFuzz* sometimes has relatively lower initial accuracy when the models are retrained. With the increase of epochs, the accuracy of the models increases rapidly, and the final accuracy based on *CAGFuzz* is mostly higher than that based on the other approaches.

▷ *Answer to* ***RQ3****:* Although different DNN have diverse performances, in general, the accuracy of the DNN can be improved by retraining them with the adversarial examples generated by *CAGFuzz*, which is reflected in the retrained model at the same time.

4.3 Threats to Validity

In the design of this study, there are several threats. In the following, we describe the main threats to validity of our approach in detail.

Internal validity: The dataset 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 division, we ask three observers to randomly exchange the examples of the two data domains. The exchanges are performed independently.

External validity: During the experimental process, the number of classification labels of the experimental datasets is limited, which may lead to the reduction of the generalisation ability of the approach. To solve this problem, we use a cross-dataset approach to validate the generalization performance of the approach across multiple datasets.

Conclusion validity: According to the designed experiments, our approach can be validated. To further ensure the validity of the conclusion, we validate the conclusion through the CIFAR-100 dataset and the models from the other researchers, and reach the same conclusion as it based on the standard**(what is standard?)** dataset.

5 Related Work

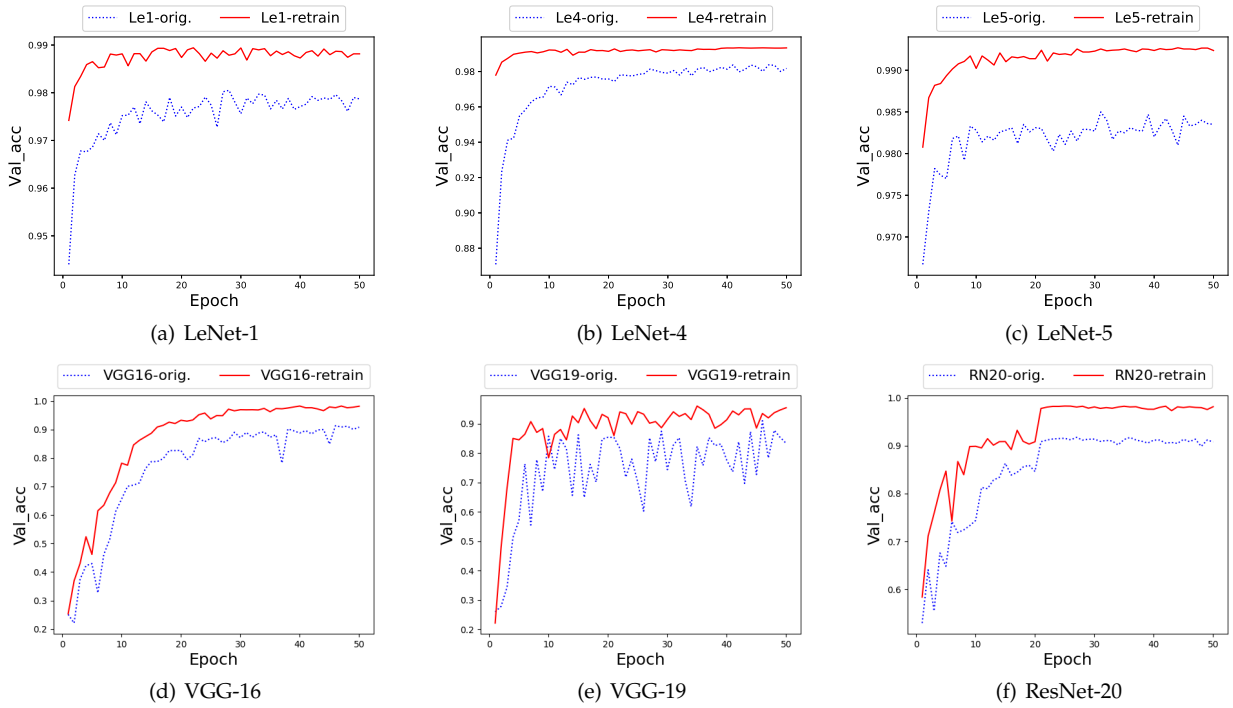
In this section, We review the most relevant work in the following three aspects: adversarial examples generation, coverage-guided fuzz testing and testing approaches of DL systems.

5.1 Adversarial Deep Learning

A large amount of research has shown that adversarial examples with small perturbations poses a great threat to the security and robustness of DL systems [36], [37], [38], [39], [40], [41]. Small perturbations added to the input images can fool the whole DL systems, where the input images are initially classified correctly by the DL systems. Whereas, the modified adversarial examples are obviously indistinguishable from the original examples in human eyes. Goodfellow et al. [15] proposed *FGSM* which can craft adversarial examples using loss function with respect to the input feature vector. Papernot et al. [42] designed JSMA to craft adversarial examples based on a precise understanding of the mapping between inputs and outputs of DNN. Kurakin et al. [43] devised BIM. They applied it multiple times with small step size, and clipped pixel values of intermediate results after each step to ensure that they are in an ϵ-neighbourhood of the original image. Carlini et al. [44] introduced CW, a new optimization-based attack technique which is arguably the most effective in terms of the adversarial success rates achieved with minimal perturbations. 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 DNN, and the generated adversarial examples have unsatisfied transferability. 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) [45], [46] is a mature defect and vulnerability detection technique. Zest [47] and



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

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

Fig. 8: Validation accuracy contrast diagram of each model in the training process. (a) LeNet-1, training on MNIST dataset, when epoch=50, (b) LeNet-4, training on MNIST dataset, when epoch=50, (c) LeNet-5, training on MNIST dataset, when epoch=50, (d) VGG-16, training on CIFAR-10 dataset, when epoch=50, (e) VGG-19, training on CIFAR-10 dataset, when epoch=50, (f) ResNet-20, training on CIFAR-10 dataset, when epoch=70.

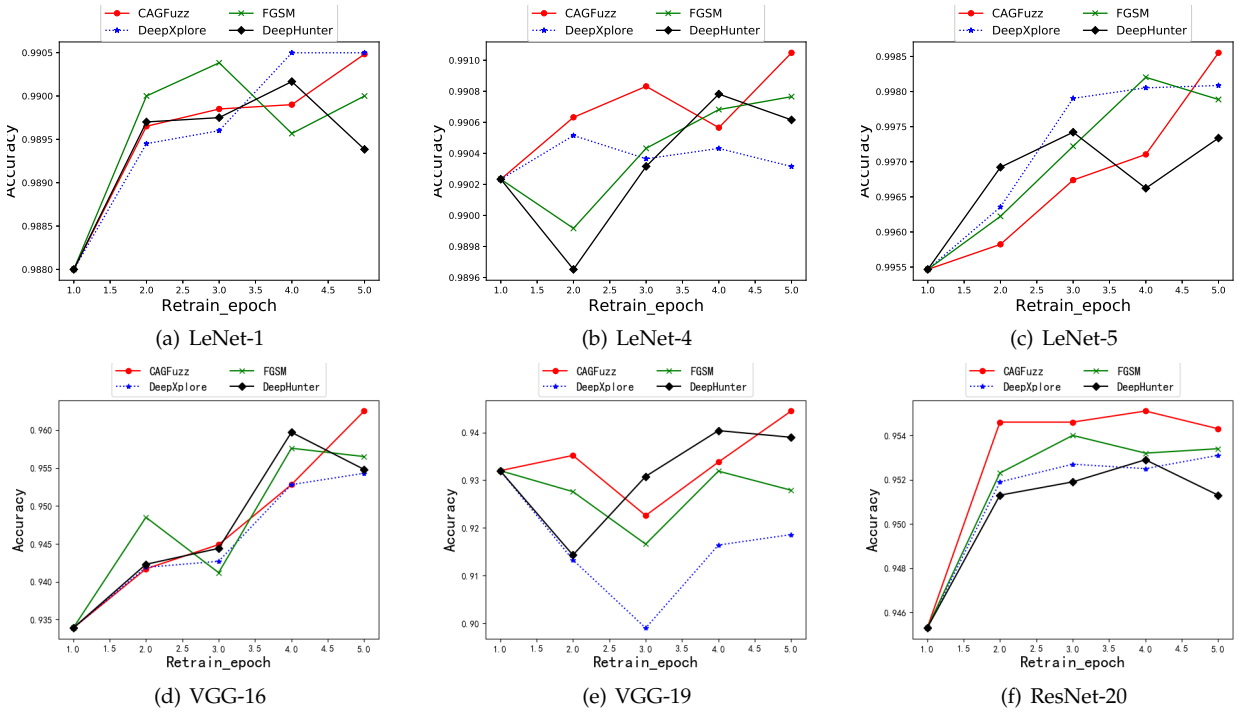
libprotobuf mutator [48] have been proposed to improve the mutation quality by providing structure aware mutation strategies. *TensorFuzz* [49] is good at automatically discovering errors that result from only a few examples. The validation of *DLFuzz* [10] shows that it is feasible to apply the fuzzy knowledge to DL testing, which can greatly improve the performance of existing DL testing technologies. Due to the inherent difference between DL systems and traditional software, traditional CGF cannot be directly applied to DL systems [50]. In our approach, CGF is adopted to adapt to DL systems. The state-of-the-art CGF mainly consists of three parts: mutation, feedback guidance, and fuzzing strategy, in which we replace the 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 fed with the same format of images, the adversarial examples generated with higher coverage are selected and added into the processing pool to maximize the neuron coverage of the target DL systems.

5.3 DL Testing Systems

In traditional software testing, the main means of machine learning system and deep learning system evaluation is to randomly extract adversarial examples from manually labeled datasets [51] and hoc simulations [52] to measure their accuracy. In some special cases, such as autopilot, special non-guided simulations are used. However, without understanding the internal mechanism of the models to be tested, such black-box test paradigms cannot find different situations that may lead to error behaviors in the future [8], [53]. *DeepXplore* [8] proposes a white-box differential testing technique for generating test inputs that may trigger inconsistencies between different DNN, which may identify incorrect behaviors. *DeepHunter* [9] performs mutations to generate new semantic retention tests, and uses multiple pluggable coverage criteria as feedback to guide test generation from different perspectives. *DeepTest* [14] runs a tool for automated testing of DNN-driven autonomous cars. In addition, many testing approaches [51], [52], [53], [54] for traditional software have also been adopted and applied to testing DL systems, such as MC/DC coverage [11], concolic test [55], combinatorial test [56] and mutation test [57]. Furthermore, various forms of neuron coverage [12] 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

We design and implement *CAGFuzz*, a coverage-guided adversarial generative fuzzing auto-testing approach for deep learning systems. *CAGFuzz* trains an adversarial example generator for a specified dataset. It generates adversarial examples for target DNN by iteratively taking original examples, and finds potential errors in the development and deployment phase of DNN. We have done extensive experiments to prove the effectiveness of *CAGFuzz* in discovering potential errors in DNN and improving model accuracy. Although neuron coverage is a disputable metric,



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

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

Fig. 9: Accuracy variation of DNN models when training set is augmented with the same number of inputs generated by different approaches. (a) LeNet-1, training on MNIST dataset, when epoch=5, (b) LeNet-4, training on MNIST dataset, when epoch=5, (c) LeNet-5, training on MNIST dataset, when epoch=5, (d) VGG-16, training on CIFAR-10 dataset, when epoch=5, (e) VGG-19, training on CIFAR-10 dataset, when epoch=5, (f) ResNet-20, training on CIFAR-10 dataset, when epoch=5.

our experimental results show that *CAGFuzz* guided by neuron coverage can discover more potential errors in different kinds of DNN models, improve their model accuracy, and further enhance the transferability of the generated adversarial samples.

For future work, we plan to employ multidimensional coverage feedback to improve the scope of the information that adversarial examples can cover. We also try to train effective adversarial example generators for other input forms, such as text information and voice information.

ACKNOWLEDGMENTS

The work is supported by the National Natural Science Foundation of China (61572171,61702159), the Natural Science Foundation of Jiangsu Province (BK20191297) and the Fundamental Research Funds for the Central Universities of China (B210202075).

REFERENCES

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

[2] A. Stocco, M. Weiss, M. Calzana, and P. Tonella, "Misbehaviour prediction for autonomous driving systems," *arXiv preprint arXiv:1910.04443*, 2019.

[3] 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.

[4] 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.

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

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

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

[12] 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.

[13] Z. Wang, M. Yan, J. Chen, S. Liu, and D. Zhang, "Deep learning library testing via effective model generation," in *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020* (P. Devanbu, M. B. Cohen, and T. Zimmermann, eds.), pp. 788–799, ACM, 2020.

[14] 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.

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

[16] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," 2014.

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

[22] X. Xie, H. Chen, Y. Li, L. Ma, Y. Liu, and J. Zhao, "Coverage-guided fuzzing for feedforward neural networks," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 1162–1165, IEEE, 2019.

[23] M. Zalewski, "American fuzzy lop," *URL: http://lcamtuf. coredump. cx/afl*, 2017.

[24] K. Serebryany, "libfuzzer a library for coverage-guided fuzz testing," *LLVM project*, 2015.

[25] 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.

[26] 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.

[27] L. Zheng, Y. Zhao, S. Wang, J. Wang, and Q. Tian, "Good practice in cnn feature transfer," *arXiv preprint arXiv:1604.00133*, 2016.

[28] 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.

[29] 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.

[30] X. Hou, J. Duan, and G. Qiu, "Deep feature consistent deep image transformations: Downscaling, decolorization and hdr tone mapping," *arXiv preprint arXiv:1707.09482*, 2017.

[31] F. Salfner, M. Lenk, and M. Malek, "A survey of online failure prediction methods," *ACM Computing Surveys (CSUR)*, vol. 42, no. 3, pp. 1–42, 2010.

[32] F. Chollet *et al.*, "Keras applications," 2018.

[33] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *arXiv preprint arXiv:1906.10742*, 2019.

[34] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," *arXiv preprint arXiv:1611.01236*, 2016.

[35] A. Kurakin, I. Goodfellow, S. Bengio, *et al.*, "Adversarial examples in the physical world," 2016.

[36] 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.

[37] Z. Zhao, D. Dua, and S. Singh, "Generating natural adversarial examples," *arXiv preprint arXiv:1710.11342*, 2017.

[38] P. Samangouei, M. Kabkab, and R. Chellappa, "Defense-gan: Protecting classifiers against adversarial attacks using generative models," *arXiv preprint arXiv:1805.06605*, 2018.

[39] 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.

[40] P. Zhang, J. Wang, J. Sun, G. Dong, X. Wang, X. Wang, J. S. Dong, and T. Dai, "White-box fairness testing through adversarial sampling," in *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020* (G. Rothermel and D. Bae, eds.), pp. 949–960, ACM, 2020.

[41] S. A. Seshia, S. Jha, and T. Dreossi, "Semantic adversarial deep learning," *IEEE Des. Test*, vol. 37, no. 2, pp. 8–18, 2020.

[42] 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.

[43] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," *arXiv preprint arXiv:1607.02533*, 2016.

[44] 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.

[45] 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.

[46] Y. Chen, B. Xuan, C. M. Poskitt, J. Sun, and F. Zhang, "Active fuzzing for testing and securing cyber-physical systems," in *ISSTA '20: 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, USA, July 18-22, 2020* (S. Khurshid and C. S. Pasareanu, eds.), pp. 14–26, ACM, 2020.

[47] 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.

[48] K. Serebryany, V. Buka, and M. Morehouse, "Structure-aware fuzzing for clang and llvm with libprotobuf-mutator," 2017.

[49] A. Odena and I. Goodfellow, "Tensorfuzz: Debugging neural networks with coverage-guided fuzzing," *arXiv preprint arXiv:1807.10875*, 2018.

[50] X. Gao, R. K. Saha, M. R. Prasad, and A. Roychoudhury, "Fuzz testing based data augmentation to improve robustness of deep neural networks,"

[51] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[52] D. L. Rosenband, "Inside waymo's self-driving car: My favorite transistors," in *2017 Symposium on VLSI Circuits*, pp. C20–C22, IEEE, 2017.

[53] I. Goodfellow and N. Papernot, "The challenge of verification and testing of machine learning," *Cleverhans-blog*, 2017.

[54] S. Gerasimou, H. F. Eniser, A. Sen, and A. Cakan, "Importance-driven deep learning system testing," in *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020* (G. Rothermel and D. Bae, eds.), pp. 702–713, ACM, 2020.

[55] 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.

[56] 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.

[57] 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.