# DLFuzz: Differential Fuzzing Testing of Deep Learning Systems

Jianmin Guo*
School of Software, Tsinghua University
Beijing, China
guojm17@mails.tsinghua.edu.cn

Yu Jiang
School of Software, Tsinghua University
Beijing, China
jiangyu198964@126.com

Yue Zhao
School of Software, Tsinghua University
Beijing, China
zhao-y17@mails.tsinghua.edu.cn

Quan Chen
Department of Computer Science and Engineering, Shanghai Jiao Tong University
Shanghai, China
chen-quan@cs.sjtu.edu.cn

Jiaguang Sun
School of Software, Tsinghua University
Beijing, China
sunjg@tsinghua.edu.cn

## ABSTRACT

Deep learning (DL) systems are increasingly applied to safety-critical domains such as autonomous driving cars. It is of significant importance to ensure the reliability and robustness of DL systems. Existing testing methodologies always fail to include rare inputs in the testing dataset and exhibit low neuron coverage.

In this paper, we propose DLFuzz, the first differential fuzzing testing framework to guide DL systems exposing incorrect behaviors. DLFuzz keeps minutely mutating the input to maximize the neuron coverage and the prediction difference between the original input and the mutated input, without manual labeling effort or cross-referencing oracles from other DL systems with the same functionality. We present empirical evaluations on two well-known datasets to demonstrate its efficiency. Compared with DeepXplore, the state-of-the-art DL whitebox testing framework, DLFuzz does not require extra efforts to find similar functional DL systems for cross-referencing check, but could generate 338.59% more adversarial inputs with 89.82% smaller perturbations, averagely obtain 2.86% higher neuron coverage, and save 20.11% time consumption.

## CCS CONCEPTS

• **Software and its engineering** → **Software testing and debugging**;

## KEYWORDS

Fuzzing Testing, Deep Learning, Neuron Coverage

**ACM Reference Format:**
Jianmin Guo, Yu Jiang, Yue Zhao, Quan Chen, and Jiaguang Sun. 2018. DLFuzz: Differential Fuzzing Testing of Deep Learning Systems. In *Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '18), November 4–9, 2018, Lake Buena Vista, FL, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3236024.3264835

*Also with Key Laboratory for Information System Security, Ministry of Education, Beijing National Research Center for Information Science and Technology (BNRist).

## 1 INTRODUCTION

In the past few years, deep learning (DL) systems have demonstrated its competitiveness on a wide range of applications, such as image classification [5, 8], natural language processing [15] and even reconstruction of brain circuits [6]. These encouraging accomplishments inspired wide deployments of DL systems in safety-critical domains, such as autonomous driving [1] and malware detection [20]. Therefore, it is in great demand to test and improve the robustness of DL systems.

For DL testing, the classical approach is to gather sufficient manually labeled data to assess the accuracy of DL systems. However, the input space of testing is so huge that it is extremely hard to collect all the possible inputs to trigger every feasible logic. It is demonstrated that state-of-the-art DL systems can be fooled by adding small perturbations to the test inputs [16], as shown in Figure 1. Therefore, DL testing is quite challenging but essential to ensure the correctness of those safety-critical practices.

Several approaches have been proposed to improve the testing efficiency of DL systems with software testing techniques. Some of them leverage solvers like Z3 to generate adversarial inputs under the formalized constraints of the DL models [4, 7]. These techniques are accurate, but work in a heavy whitebox manner and are resource-consuming for constraint solving. Some blackbox methods exploit heuristic algorithms to mutate the inputs until the adversarial inputs acquired [19]. These methods are time-consuming and rely heavily on the manually supplied ground truth labels. Recently, DeepXplore [13] was presented as the state-of-the-art whitebox testing framework for DL systems and first introduced the concept of neuron coverage as a testing metric. Meanwhile, it requires multiple DL systems with similar functionality as cross-referencing oracles to avoid manual checking. Nevertheless, cross-referencing suffers from the scalability and difficulty of finding similar DL systems. Other approaches of adversarial deep learning focus on fooling the DL systems by applying imperceptible perturbations to the inputs mostly in a gradient-based manner [12, 16]. They work efficiently but are shown to have low neuron coverage [13].

In this paper, we propose DLFuzz[1], the first differential fuzzing testing framework. It aims to maximize the neuron coverage and

[1]https://github.com/turned2670/DLFuzz

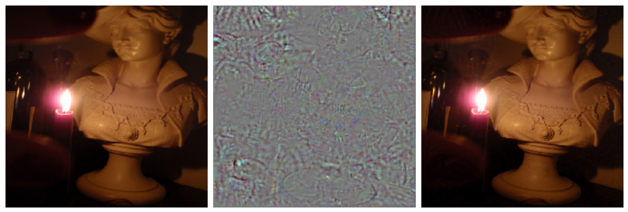

**Figure 1: An adversarial input for DL model VGG-16[14] generated by DLFuzz. (Left) is correctly predicted as "candle", (center) perturbation applied to the left to obtain the right, (right) adversarial input predicted as "lampshade".**

generate more adversarial inputs for a given DL system, without cross-referencing other DL systems or manual labeling. First, DLFuzz iteratively selects neurons worthy to activate for covering more logic, and mutates the test inputs by applying minute perturbations. During the mutation process, DLFuzz keeps the mutated inputs which contribute to a certain increase of the neuron coverage for the subsequent fuzzing. The minute perturbation is restricted to be invisible and ensures the prediction results before and after the mutation to be the same. In this way, DLFuzz is able to obtain rare inputs, and automatically identify the erroneous behaviors with differential testing that an error is triggered when the prediction result of the mutated input is not the same with the original input.

To evaluate the efficiency of DLFuzz, we conducted empirical studies on six DL systems trained on two popular datasets, MNIST [9] and ImageNet [3]. The DL systems and the datasets are exactly the same as those used by DeepXplore. Compared with DeepXplore, DLFuzz does not need extra efforts to collect similar functional DL systems for cross-referencing label check, but could generate 135% – 584.62% more adversarial inputs with 79.56% – 96.77% smaller perturbations, and obtain 1.10% – 5.59% higher neuron coverage. For the time efficiency, it saves 20.11% time consumption in average with one exceptional case on ImageNet costing 59.42% more than DeepXplore.

## 2 MOTIVATION

There exists a large gap between DL testing and traditional software testing, owing to the totally distinct internal structures of deep neural networks (DNN) and software programs. We aim to break the resource consumption limitation of blackbox testing techniques[19] and the cross-referencing obstacles of whitebox testing techniques[13].

Fuzzing testing [2, 11, 18, 21] has been recognized as one of the most effective methodologies for vulnerability detection in software testing, demonstrated by the huge amount of vulnerabilities caught. The core idea is to generate random inputs to execute as many program paths as possible so as to lead the program to expose violations and crashes. It can be seen that fuzzing testing and DL testing share similar goals of achieving higher coverage as well as getting more exceptional behaviors. In general, we combined the knowledge in key stages of fuzzing into DL testing as below:

(1) **Optimization Goal.** The goal of reaching higher neuron coverage and exposing more exceptional behaviors can be treated as a joint optimization problem. This optimization problem can be implemented in the gradient-based manner.
(2) **Seed Maintenance.** While fuzzing, the mutated inputs which contribute to a certain increase of the neuron coverage are kept in the seed list, based on the potential to improve neuron coverage continuously in the subsequent fuzzing.
(3) **Diversity in Mutation Strategies.** We designed many neuron selection strategies to select neurons that are possible to cover more logic and trigger more incorrect outputs. Furthermore, multiple mutation ways for test inputs have been already practiced, and are easy to be integrated.

## 3 DLFUZZ APPROACH

### 3.1 Architecture

The overall architecture of DLFuzz is depicted in Figure 2. In this paper, we implement DLFuzz to work on image classification, a popular task in DL domains to demonstrate its feasibility and effectiveness. The adaptions in other tasks such as speech recognition are straightforward and also follow the same workflow in Figure 2.

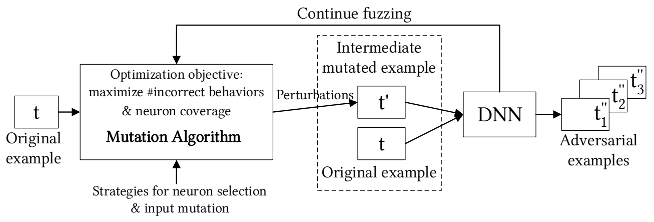
Continue fuzzing
t
Original example
Optimization objective: maximize #incorrect behaviors & neuron coverage
Mutation Algorithm
Strategies for neuron selection & input mutation
Perturbations
Intermediate mutated example
t'
t
Original example
DNN
t''_1
t''_2
t''_3
Adversarial examples

**Figure 2: Architecture of DLFuzz**

To specify, the given test input $t$ is an image to be classified, the DNN is a particular convolutional neural network (CNN) under test, such as VGG-16 [14]. The mutation algorithm applies tiny perturbation to $t$ and gets $t'$, which is visibly indistinguishable from $t$. If the mutated input $t'$ and the original input $t$ are both fed to the CNN but classified to be of different class labels, we treat this as an incorrect behavior and $t'$ to be one of the adversarial inputs. The inconsistent classification results indicate that at least one of them is wrong so that manually labeling effort is not required here. In contrast, if the two are predicted of the same class label, $t'$ will continue to be mutated to test the CNN's robustness.

### 3.2 Algorithm

The mutation algorithm is the main component of DLFuzz. It is completed by solving a joint optimization problem of both maximizing the neuron coverage and the number of incorrect behaviors. Based on the demonstration that covering more neurons could potentially trigger more logic and more erroneous behaviors [13], DLFuzz also leverages the same definition and computing way of neuron coverage as DeepXplore [13]. Neurons with output values larger than the set threshold are regarded as activated (covered).

The core process of the mutation algorithm is in Algorithm 1. The algorithm contains three key components to discuss in detail.

**Optimization Problem.** As discussed in Section 1, the gradient-based adversarial deep learning outperforms the other approaches in several aspects, especially in time efficiency. It founds perturbations by optimizing the input to maximize the prediction error [16], which is opposite to optimizing the weights to minimize the prediction error while training the DNN. It is easy to implement by

**Algorithm 1** Mutation Algorithm

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

customizing the loss function as our objective and maximizing the loss by gradient ascent.

The loss function of DLFuzz is defined as the following equation (Algorithm 1 line 7), which is also the optimization objective:

$$obj = \sum_{i=0}^{k} c_i - c + \lambda \cdot \sum_{i=0}^{m} n_i \tag{1}$$

The objective consists of two parts. In the first part $\sum_{i=0}^{k} c_i - c$, $c$ is the original class label of the input, $c_i (i = 0, ..., k)$ is one of top k class labels with confidence just lower than $c$. Maximizing the first part guides the input to cross the decision boundary of the original class and lie in the decision space of top k other classes. Such modified inputs are more likely to be classified incorrectly [12]. In the second part $\sum_{i=0}^{m} n_i$, $n_i$ is a target neuron intended to activate. These neurons are selected considering many strategies to improve neuron coverage (Algorithm 1 line 6). The hyperparameter $\lambda$ is used for balancing the two objectives.

**Fuzzing Process.** The fuzzing process reveals the overall workflow of Algorithm 1. When given a test input $x$, DLFuzz maintains a seed list for keeping the intermediate mutated inputs which contribute to neuron coverage. Initially, the seed list only has one input which is exactly $x$. Next, DLFuzz traverses each seed $x_s$ and obtains the elements making up its optimization objective. Then, DLFuzz computes the gradient direction for later mutation. In the mutation process, DLFuzz iteratively applies the processed gradient as the perturbation to $x_s$ and obtains the intermediate input $x'$. After each mutation, the intermediate class label $c'$, coverage information, the metric $l_2$_distance are acquired. If the neuron coverage improved by $x'$ and $l_2$_distance are desired, $x'$ will be added into the seed list. Finally, if $c'$ is already different from $c$, mutation process for seed $x_s$ terminates and $x'$ will be included in the set of adversarial inputs. Therefore, DLFuzz is able to generate multiple adversarial inputs for a certain original input and explore a new way to further improve neuron coverage.

For the iterative mutation process, first, various processing methods are available to generate perturbations when the gradients obtained, including just keeping the sign [16], imitating the realistic situations [13, 17], etc. These mutation strategies for the input are easy to be extended to DLFuzz. Second, DLFuzz adopts $l_2$ distance to measure the perturbation with similar computation with [12]. As for the conditions of seed keeping in line 16, DLFuzz limits our desired distance to a relatively small range (less than 0.02) to ensure the imperceptibility. As the neuron coverage improvement of one input declines with time, the corresponding threshold for keeping the seed also decreases with running time.

**Strategies for Neuron Selection.** To maximize neuron coverage, we propose four heuristic strategies for selecting neurons more likely to improve coverage. For each seed $x_s$, $m$ neurons will be selected utilizing one or multiple strategies, which can be customized in *strategies* of the algorithm inputs.

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

## 4 EXPERIMENT

### 4.1 Experiment Setup

We implemented DLFuzz based on the widespread frameworks, Tensorflow 1.2.1 and Keras 2.1.3. Tensorflow and Keras provide the efficient interfaces for computing the gradients and recording the intermediate output of all neurons after each prediction of the DNN. We developed and evaluated DLFuzz on a computer with 4 cores (Intel i7-7700HQ @3.6GHz), 16GB of memory, a NVIDIA GTX 1070 GPU and Ubuntu 16.04.4 as the host OS.

For evalution, we selected two datasets (MNIST and ImageNet) and the corresponding CNNs used by DeepXplore for image classification tasks. MNIST [9] is a large database of handwritten digits consisting of 70000 images. ImageNet [3] is a large visual database containing over 14 million images for object recognition. The same as DeepXplore, DLFuzz tested three pre-trained models for each dataset, that is, LeNet-1 [10], LeNet-4 [10], LeNet-5 [10] for MNIST and VGG-16 [14], VGG-19 [14], ResNet50 [5] for ImageNet. Considering the fairness, we also randomly choose 20 input images from the dataset for each CNN in the same way with DeepXplore. If not specified, the default settings of parameters $k$, $m$, $strategies$, $iter\_times$ are $4, 10,$ "$strategy$ 1", 3 respectively for their overall good performance.

## 4.2 Result

Table 1 presents the effectiveness of DLFuzz compared with DeepXplore. DLFuzz exhibits its advantages in improving neuron coverage, generating more adversarial inputs within the same time limit and restricting imperceptible perturbations. First, as presented in the third column of the table, for the tested six CNNs, DLFuzz achieves 1.10% to 5.59% higher neuron coverage than DeepXplore in different settings in average. For the best setting, DLFuzz is able to acquire 13.42% higher neuron coverage.

**Table 1: Effectiveness of DLFuzz compared with DeepXplore.**

| DataSet | Model (#Neurons) | NC Imp.[1] | $l_2$ Distance | #Adv.[2] | Adv. Time[3] |
|---|---|---|---|---|---|
| MNIST | LeNet-1(52) | 2.45% | 8.2637/0.2708 | 20/53 | 0.7078/0.5623 |
| | LeNet-4(148) | 5.59% | 8.2637/0.2812 | 20/47 | 0.7078/0.6344 |
| | LeNet-5(268) | 2.23% | 8.2637/0.2670 | 20/54 | 0.7078/0.5870 |
| ImageNet | VGG16(14888) | 3.52% | 0.0817/0.0167 | 13/89 | 10.473/3.4537 |
| | VGG19(16168) | 2.28% | 0.0817/0.0154 | 13/81 | 10.473/3.6606 |
| | ResNet50(94056) | 1.10% | 0.0817/0.0097 | 13/72 | 10.473/16.6958 |

Comparisons represented by content in format a/b, where a denotes the result of DeepXplore and b denotes the result of DLFuzz.
[1] Average neuron coverage improvement.
[2] Number of adversarial inputs generated.
[3] Average time of generating per adversarial input.

Next, adversarial inputs generated by DLFuzz have much smaller perturbations. As the cases in Figure 3, the perturbations generated by DeepXplore are visible while those generated by DLFuzz are invisible and imperceptible. In this way, DLFuzz provides stronger guarantee for the consistence of the image's identity before and after mutation. As shown in the fourth and fifth columns of Table 1, DLFuzz averagely generated 338.59% more adversarial inputs with 89.82% smaller perturbations. Moreover, DLFuzz spent 20.11% shorter time on generating each adversarial input on these DL systems. An exceptional case is that DLFuzz cost more time on generating adversarial inputs than DeepXplore for ResNet50, which is owing to more time needed for neuron selection when testing a DL system consisting of a huge amount of neurons (94056).

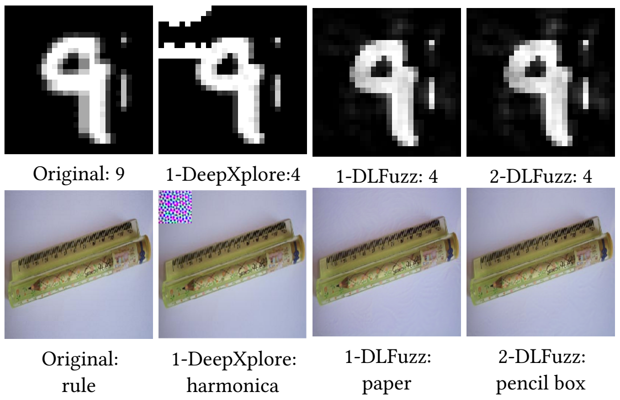

**Figure 3: Cases of adversarial inputs annotated with the framework and the predicted label. The above row for MNIST and the below for ImageNet.**

We also tried all the proposed neuron selection strategies on two CNNs and depicted the results in Figure 4. All strategies are shown to contribute more to neuron coverage improvement than DeepXplore while have similar performance among themselves. It seems that "strategy 1" performs slightly better. In addition, we incorporated 114 adversarial images into the training set of three CNNs on MNIST and retrained them trying to increase their accuracy. Finally, we improve their accuracy by up to 1.8% within 5 epochs. More improvement is expected if more adversarial inputs included in the retraining process.

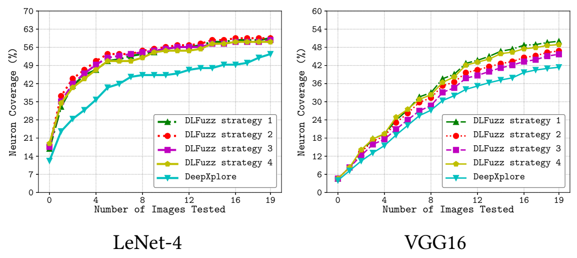

**Figure 4: Neuron coverage with number of images tested when different strategies applied in DLFuzz.**

## 4.3 Discussion

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

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

## 5 CONCLUSION

We design and implement DLFuzz as an effective fuzzing testing framework of DL systems. DLFuzz first combines the basic ideas of fuzzing testing into DL testing and demonstrates its efficiency. Compared with DeepXplore, DLFuzz averagely obtained 2.86% higher neuron coverage and generated 338.59% more adversarial examples with 89.82% smaller perturbations given the same amount of inputs. DLFuzz also overcomes the trouble of relying on multiple DL systems of the similar functionality in DeepXplore. Additionally, DLFuzz exhibits its practical use by incorporating these adversarial inputs to retrain the DL systems and to steadily improve their accuracy.

## REFERENCES

[1] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* (2016).

[2] Yuanliang Chen, Yu Jiang, Jie Liang, Mingzhe Wang, and Xun Jiao. 2018. EnFuzz: From Ensemble Learning to Ensemble Fuzzing. *arXiv preprint arXiv:1807.00182* (2018).

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on.* IEEE, 248–255.

[4] Shixiang Gu and Luca Rigazio. 2014. Towards deep neural network architectures robust to adversarial examples. *arXiv preprint arXiv:1412.5068* (2014).

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 770–778.

[6] Moritz Helmstaedter, Kevin L Briggman, Srinivas C Turaga, Viren Jain, H Sebastian Seung, and Winfried Denk. 2013. Connectomic reconstruction of the inner plexiform layer in the mouse retina. *Nature* 500, 7461 (2013), 168.

[7] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. 2017. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification.* Springer, 3–29.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems.* 1097–1105.

[9] Yann LeCun. 1998. The MNIST database of handwritten digits. *http://yann. lecun. com/exdb/mnist/* (1998).

[10] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[11] Jie Liang, Mingzhe Wang, Yuanliang Chen, Yu Jiang, and Renwei Zhang. 2018. Fuzz testing in practice: Obstacles and solutions. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER).* IEEE, 562–566.

[12] Seyed Mohsen Moosavi Dezfooli, Alhussein Fawzi, and Pascal Frossard. 2016. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[13] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Deepxplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles.* ACM, 1–18.

[14] Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations.*

[15] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems.* 3104–3112.

[16] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013). https://arxiv.org/abs/1312.6199

[17] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th International Conference on Software Engineering.* ACM, 303–314.

[18] Mingzhe Wang, Jie Liang, Yuanliang Chen, Yu Jiang, Xun Jiao, Han Liu, Xibin Zhao, and Jiaguang Sun. 2018. SAFL: increasing and accelerating testing coverage with symbolic execution and guided fuzzing. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings.* ACM, 61–64.

[19] Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. 2018. Feature-guided black-box safety testing of deep neural networks. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems.* Springer, 408–426.

[20] Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, and Yibo Xue. 2014. Droid-sec: deep learning in android malware detection. In *ACM SIGCOMM Computer Communication Review*, Vol. 44. ACM, 371–372.

[21] Michal Zalewski. 2007. American fuzzy lop.