Available online at www.ijpe-online.com

International Journal of Performability Engineering

vol. 15, no. 10, October 2019, pp. 2675-2682
DOI: 10.23940/ijpe.19.10.p13.26752682

FDFuzz: Applying Feature Detection to Fuzz Deep Learning Systems

Jie Wang[a,b,*], Kefan Cao[b], Chunrong Fang[b], and Jinxin Chen[b]

[a]*Maanshan Teacher's College, Maanshan, 243000, China*
[b]*The State Key Laboratory of Novel Software Technology, Nanjing University, Nanjing, 210000, China*

Abstract

In the past years, many resources have been allocated to research on deep learning networks for better classification and recognition. These models have higher accuracy and wider application contexts, but the weakness of easily being attacked by adversarial examples has raised our concern. It is widely acknowledged that the reliability of many safety-critical systems must be confirmed. However, not all systems have sufficient robustness, which makes it necessary to test these models before going into service. In this work, we introduce FDFuzz, an automated fuzzing technique that exposes incorrect behaviors of neural networks. Under the guidance of the neuron coverage metric, the fuzzing process aims to find those examples to let the network make mistakes via mutating inputs, which are then correctly classified. FDFuzz employs a feature detection technique to analyze input images and improve the efficiency of mutation by features of keypoints. Compared with TensorFuzz, the state-of-the-art open source library for neural network testing, FDFuzz demonstrates higher efficiency in generating adversarial examples and makes better use of elements in corpus. Although our mutation function consumes more time to generate new elements, it can generate 250% more adversarial examples and save testing time.

Keywords: neuron network; machine learning; adversarial examples; fuzzing

(Submitted on August 8, 2019; Revised on September 12, 2019; Accepted on October 20, 2019)

1. Introduction

Deep learning was first proposed as machine learning in 1986 [1], when Hinton et al. applied the backpropagation algorithm into multilayer perceptron [2] and regained the attention to AI. Immediately after, LeNet [3] was created, and it achieved great success in handwritten number recognition. Since 2000, deep learning has normally been called artificial neural networks.

After many years of research, a large number of excellent deep learning models have been established for different contexts, including face recognition, medical diagnosis, natural language processing, autonomous driving, and smart contracts. Compared with traditional models, neural networks perform much better in many aspects, including perception, learning, and cognition. Thus, these models have more potential to do well in complex tasks, but they do have some disadvantages. One of the most severe drawbacks is a lack of robustness, which means systems using deep learning models could be fooled by inputs triggering wrong logic. Unfortunately, these mistakes cannot be ignored, especially in safety-critical tasks like automated driving. If a traffic light is misclassified, the safety of passengers could be threatened.

Therefore, the robustness of deep learning models needs to be tested and improved. Inputs leading to the misclassification of deep learning systems with imperceptible mutation of original inputs are called adversarial examples. However, adversarial examples have a large input space, and the principle of triggering incorrect classification cannot be easily explained, thus making it hard to generate them manually. Considering all the difficulties of searching adversarial examples, we need to leverage testing techniques to improve the generation efficiency and find more of them.

Several approaches have been proposed to test deep learning systems. TensorFuzz[4] is a state-of-the-art open source framework for neural network testing. It can be used to test most neural networks with quick mutation and coverage analysis,

* Corresponding author.
E-mail address: star 502@126.com

but its efficiency is not good; it simply adds white noise to elements in a corpus. Its coverage analysis focuses only on the output and ignores other parts of the deep learning system. DLFuzz [5] leverages the neuron activation ratio to solve the feedback information and calculate the coverage. Its coverage function is much more complicated than simply obtaining the output summation of the neuron network, but it considers system structure more and performs better in practice. The mutation process of DLFuzz adds white noise to original inputs just like TensorFuzz, which is quick, but it lacks the guidance of generating new elements and thereby reduces the efficiency. Feature-guided black-box safety testing [6] uses the scale invariant feature transform (SIFT) algorithm to guide the process of mutation with features of images. However, the Monte Carlo tree search ignores the structure of neural network models, and consumes much more time to obtain an adversarial example than fuzzing.

In our work, we propose FDFuzz, a coverage guided fuzzing technique with feature detection mutation process and neuron coverage analysis. To evaluate the efficiency of FDFuzz, we choose a LeNet model with 99% accuracy to be tested. With the same selected images, our FDFuzz can generate many more adversarial examples than TensorFuzz, which means generating each example takes slightly more time.

In particular, we make the following contributions:

- Aiming to find adversarial examples of deep learning systems, we propose an effective coverage-guided fuzzing technique called FDFuzz.
- Our work leverage feature detection and neuron coverage analysis improve the efficiency of the testing process.
- We prove that the mutation or coverage process in fuzzing that is simple and cheap to compute may not always be the best choice.

2. Motivation

In this section, we give an overview of coverage-guided fuzzing and introduce the feature detection algorithm SIFT and neuron coverage.

2.1. Coverage-Guided Fuzzing

Fuzz testing, or fuzzing, is not a brand new technique in software testing. It has proven itself of its efficiency in searching for vulnerabilities of systems, and AFL [7] is the most famous of them.

During the testing process, seeds are sampled from valid input cases and transformed into a number of original elements. These elements are created to initialize a corpus as a collection for fuzzing. In the feedback loop, some of the elements are chosen from the corpus to be mutated for generating new inputs to the system. Regardless of whether these new inputs cause crash and errors or not, their coverage is calculated for evaluation based on the feedback information. Some of the new inputs have potential value to trigger vulnerabilities, and they will be added to the corpus for the next loop.

It is widely acknowledged that there is a large gap between deep learning systems and traditional software systems, caused by distinct structures and data formats. However, they share a similar testing target of discovering vulnerabilities of systems, which make it possible to apply the fuzzing technique to deep learning systems. Considering the implementation of deep learning systems, traditional ways of mutating and calculating coverage cannot be used directly, so we must leverage other methods as a substitution.

2.2. SIFT Algorithm

The scale invariant feature transform (SIFT) algorithm [8] is an effective and reliable computer vision technique that transforms information exhumed from an image into a set of feature vectors. These features are stable and invariant to changes of rotation, scaling, brightness, or noise. First, the Gaussian kernel is used to generate different scales of an image, simulating human perception at various angles. These scales are calculated as follows:

$$L(x, y, \sigma) = G(x, y, \sigma) \times I(x, y) \tag{1}$$

Where $G(x, y, \sigma)$ is a two-dimensional Gaussian kernel, and different values of σ represent different angles.

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{\frac{x^2+y^2}{2\sigma^2}} \quad (2)$$

Then, we compare the pixel value of the same point in different angles, together with different points within one angle, to detect the extrema areas. Some of these areas are noises, so we leverage a Hessian matrix of the intensity and Taylor expansion to remove them. After that, we can locate keypoints successfully and calculate their size and orientation. The descriptor of each keypoint finally transforms location and other information into a 128-element feature vector, indicating the gradient and size of features.

In addition to keypoints, all the pixels in an image could affect the recognition result. Our work leverages a Gaussian mixture model based on these keypoints [6], to ensure a comprehensive analysis of all pixels.

2.3. Neuron Coverage

The concept of neuron coverage was first proposed by DeepXplore [9], a well-known white-box testing framework. Rather than merely calculate the output of the final layer, neuron coverage refers to each layer of the neural network model and evaluates the output value of each neuron. A neuron is treated as activated if its value is higher than the threshold, and the coverage of the new element is calculated as the ratio of activated neurons. We leverage the same way to evaluate coverage of new elements as TensorFuzz.

3. Our Approach

Figure 1 shows the architecture of FDFuzz. We implement FDFuzz to handwrite number recognition tasks, which is a basic task to analyze the efficiency of generating adversarial examples of deep learning systems.

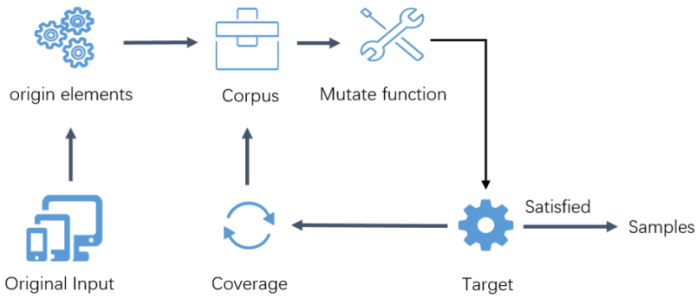

Figure 1. Architecture of FDFuzz

The original input contains images selected from the training dataset. After transforming it into a batch of original elements, we create a corpus to keep them and all other elements generated in fuzzing. The mutation function chooses some elements in the corpus each time and generates new elements by changing pixel values. Then, the elements are put into the deep learning system to check whether they are correctly recognized. The elements causing mistakes will be treated as adversarial examples. Others will be collected and analyzed by the coverage function, and their coverage determines whether they will be added into the corpus for the next loop.

3.1. Mutation

The mutation process is shown in Figure 2. An element selected from the corpus contains abundant information, including image resources, labels, and other data like coverage, and we need to obtain the image resources first for detection. Then, we apply the SIFT algorithm implemented in OpenCV[10], a well-known computer version library, to obtain keypoints of the image with the response strength. Not all these points have close response strength; some of them are so weak that they should be removed to ensure efficiency. Then, we utilize these points to fit Gaussian distribution models to mutate other pixels. The mutation degree of the pixel value is determined by the location of the pixel and response strength of keypoints.

3.2. Coverage

The neuron coverage process is shown in Figure 3. After giving a new element to the system, outputs of neurons in each layer can be fetched by the fetch function in the TensorFuzz library. We analyze output values and calculate the activation ratio as the coverage of the neural network model.

Algorithm 1 SIFT Mutation

Input:
element: element from corpus to be detected
threshold: parameter for selecting keypoints
count: number of new elements generated by mutation
Output: mutated new elements, *rets*

```
rets = []
get image resource from element
use SIFT algorithm to get keypoints
for each one in keypoints do
    if response < threshold then
        remove it from keypoints
    end if
end for
for index = 0 to count do
    decide the number of pixels to be mutated.
    for each pixel to be mutated do
        form NormalDistribution by one of keypoints
        get location of pixel
        mutate the pixel according to response strength
        add new element with mutated pixel to rets
    end for
end for
```

Figure 2. Algorithm of mutation process

Algorithm 2 Neuron Coverage

Input:
coverageBatches: neuron outputs of new elements
threshold: parameter to consider whether neurons are activated
Output: coverage of generated elements, *coverageList*

```
coverageList = []
for iter = 0 to len(coverageBatches) do
    total = 0
    activated = 0
    for each layer of model do
        Normalize outputs of neurons
        for each neuron in the layer do
            total = total + 1
            if output > threshold then
                activated = activated + 1
            end if
        end for
    end for
    rate = activated/total
    add rate to coverageList
end for
```

Figure 3. Algorithm of neuron coverage process

4. Evaluation

We finally decide to design two experiments to assess the efficiency of FDFuzz and TensorFuzz in different aspects. Experimental results can indicate that although FDFuzz needs much more time to generate each new elements, it can find more adversarial examples than TensorFuzz as complementary, which improves the overall efficiency of fuzzing as a result.

4.1. Experimental Setup

To simplify our experiment, we select MNIST[11] as our training dataset, and both experiments are based on a classic five-layer LeNet-1[3], which contains two convolution layers and two pooling layers. The structure of the LeNet-1 model to be tested is shown in Table 1. After training 100,000 steps with the Adadelta optimizer and a learning rate of 0.1, the accuracy of recognizing handwriting numbers is almost 99%.

Table 1. Structure of neuron network model

Layer name	Layer size
convolution	5×5×4
maxpooling	2×2
convolution	5×5×4
maxpooling	2×2
flatten	
dense+softmax	10

FDFuzz is implemented in Python 3.6 based on the famous framework TensorFlow, which provides convenient interfaces for model construction and access to intermediate output of neurons in the model. The evaluation process is conducted on a computer with i9-8950HK @2.90GHz, 16.0GB RAM, and an NVIDIA GTX 1080 GPU.

As we mentioned before, two experiments are prepared to evaluate the efficiency of TensorFuzz and FDFuzz in different aspects. The first experiment analyzes which can generate more adversarial examples with the same batch of original inputs, which means the seeds given to TensorFuzz and FDFuzz are the same. Another experiment shows how much time it takes for FDFuzz and TensorFuzz to generate the same number of adversarial examples, regardless of whether they are the same or not.

4.2. Experiment of Generation Efficiency

To make the process of our experiment clear, we set a restriction that the sample function must fetch only one image from MNIST each time with its label to initialize the same corpus for FDFuzz and TensorFuzz. The aim is to avoid the disturbance of batch size and content on the results of the experiment. Then, we set the total number of experiments to generate adversarial examples, so as to determine when the experiment needs to stop. It cannot be guaranteed that an adversarial example is found each time.

Figure 4 summarizes the results. The red spots indicate the time for FDFuzz to generate an adversarial example, while the green spots are related to TensorFuzz. With the total number of 50 attempts, FDFuzz generates ten more adversarial examples than TensorFuzz. To display the results more clearly, some of the examples that took significantly more time than ten seconds are not shown in Figure 1. All the details for time consumption are shown in Table 2, not including those failing to create adversarial examples. In this experiment, FDFuzz has a much higher efficiency (almost 250%) than TensorFuzz in generating adversarial examples. This indicates that our tool makes the testing process more complicated but is likely to have better performance than Tensorflow.

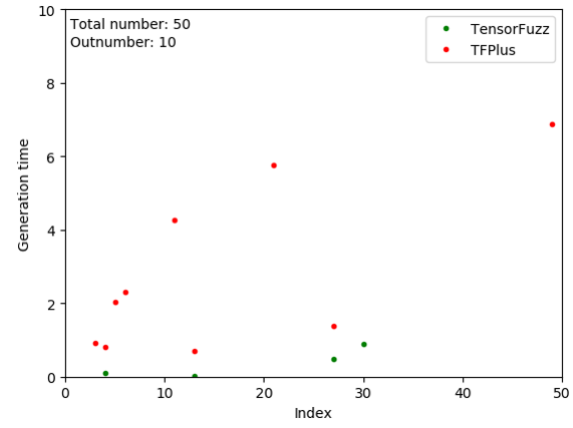

Figure 4. Comparison of numbers

4.3. Experiment of Generation Time

In this experiment, we analyze how much time is taken to generate the same number of adversarial examples with the same origin corpus. Considering the fact that FDFuzz leverages the SIFT algorithm for generation and neuron coverage for evaluation to improve the performance, it is natural to consume much more time to deal with an adversarial example. If FDFuzz can save more time in total, it is a better choice.

Table 2. Time consumption

Index	TensorFuzz	FDFuzz
2	-1	0.9208
3	0.0995	0.7994
4	-1	2.0268
5	-1	2.3008
10	-1	4.2638
12	0.0077	0.6951
14	-1	37.7725
19	-1	15.6905
20	-1	5.7677
21	-1	10.7386
26	0.3788	1.3709
29	0.8972	-1
30	-1	12.3731
38	-1	11.8466
48	-1	6.8629

As is shown in Figure 5, FDFuzz needs more time for each adversarial example. However, the advantage in quantity helps FDFuzz save a large amount of time and perform much better.

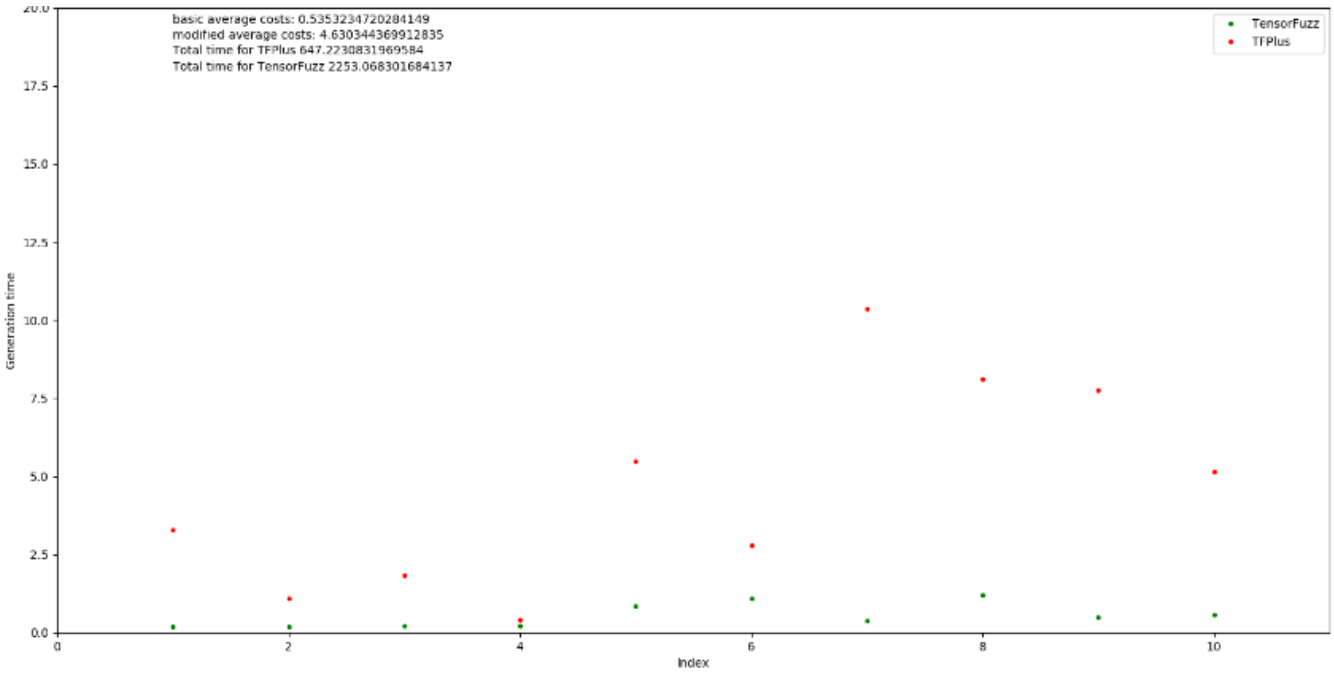

Figure 5. Comparison of Time

5. Related Work

In this section, we introduce other works related to FDFuzz. As we explained above, fuzzing technique, especially coverage-guided fuzzing, has been widely applied in traditional software testing, like AFL [7] and LibFuzzer [12]. It has been expanded to improve testing efficiency; for example, [13] leveraged a Markov chain to decide which operation of mutation should be chosen next and designed various kinds of power function to affect the number of new elements generated by mutation. FasterFuzz [22] uses Generative Adversarial Network (GAN) models to reinitialize the system with novel seed files, in order to improve the performance of AFL.

However, as mentioned in [5], there is a large gap between the implementation of traditional software and neuron network models, which makes it complicated to work without adjustment. Researchers have noticed that there may be a

chance to win again by CGF. These works cover both the white-box manners, black-box manners, and grey-box manners [2, 4-6, 9, 12-20]. Considering the different methods of testing objects and evaluation indexes, it is difficult to determine which is better. [6] used the SIFT algorithm, but it selected Monte Carlo tree search for searching. DLFuzz [5] is based on [9] with neuron coverage, and its mutation part is similar to TensorFuzz. The leverage neuron coverage in [9] and [16] dealt with feedback information, but the implementation was not the same. DeepHunter[21] presents a coverage-guided fuzz testing framework for deep neural networks, which generates new semantically preserved tests with a metamorphic mutation strategy, and guides the test generation with multiple extensible coverage criteria as feedback. Besides, DeepHunter incorporates both diversity-based and recency-based seed selection.

Thus, we believe that there is still not an existing process or model that is more efficient and accurate than other ones in all usage scenarios. Considering the different requirements and restrictions, we must select our tools and methods, and many approaches are complementary to each other in various situations.

6. Conclusions

We design and implement FDFuzz as a new fuzzing framework for deep learning systems. FDFuzz leverages the coverage-guided fuzzing technique and combines feature detection and neuron coverage. Compared with TensorFuzz, we have demonstrated its efficiency in generating and discovering adversarial examples.

In the future, we plan to explore other choices of mutation and coverage calculation. Additionally, we plan to modify the implementation of other parts in fuzzing, such as sample function and power function (dynamically determining the number of mutated elements).

Acknowledgements

This research was supported by the National Natural Science Foundation of China (No. 61802171, 61772014), the Excellent Youth Talent Support Project of University of Anhui in 2017 (No. gxyq2017242), the Fundamental Research Funds for the Central Universities (No. 021714380017), and the Open Foundation of State Key Laboratory for Novel Software Technology in Nanjing University (No. ZZKT2017B09).

References

1. M. R. Minar and J. Naher, "Recent Advances in Deep Learning: An Overview," arXiv:1807.08169, 2018
2. R. David, H. Geoffrey, and W. Ronald, "Learning Representations by Back-Propagating Errors," *Nature*, Vol. 323, pp. 533-536, 1986
3. Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based Learning Applied to Document Recognition?" in *Proceedings of the IEEE 86.11*, pp. 2278-2324, 1998
4. A. Odena and I. Goodfellow, "TensorFuzz: Debugging Neural Networks with Coverage-Guided Fuzzing," arXiv:1807.10875, 2018
5. J. Guo, Y. Jiang, Y. Zhao, Q. Chen, and J. Sun, "DLFuzz: Differential Fuzzing Testing of Deep Learning Systems," arXiv:1808.09413, 2018
6. M. Wicker, X. Huang, and M. Kwiatkowska, "Feature-Guided Black-Box Safety Testing of Deep Neural Networks," arXiv:1710.07859, 2017
7. M. Zalewski, "American Fuzzy Lop," (http://lcamtuf.coredump.cx/afl/)
8. D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, Vol. 60, No. 2, pp. 91-110, 2004
9. K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated Whitebox Testing of Deep Learning Systems," arXiv:1705.06640, 2017
10. G. Bradski, "The OpenCV Library," *Dr. Dobbs Journal of Software Tools*, pp. 120-125, 2000
11. Y. LeCun, "The MNIST Database of Handwritten Digits," (http://yann.lecun.com/exdb/mnist/, 2018)
12. K. Serebryany, "Continuous Fuzzing with Libfuzzer and Addresssanitizer?" in *Proceedings of 2016 IEEE Cybersecurity Development* (*SecDev*), pp. 157-157, 2016
13. M. Bohme, V. Pham, and A. Roychoudhury, "Coverage-based Greybox Fuzzing as Markov Chain?" in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security CCS*, pp. 1032-1043, 2016
14. S. Gu and L. Rigazio, "Towards Deep Neural Network Architectures Robust to Adversarial Examples," arXiv:1412.5068, 2014
15. X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety Verification of Deep Neural Networks?" in *Proceedings of International Conference on Computer Aided Verification*, pp. 3-29, Springer, 2017
16. L. Ma, F. Juefei-Xu, J. Sun, C. Chen, T. Su, F. Zhang, et al., "Deepgauge: Comprehensive and Multi-Granularity Testing Criteria for Gauging the Robustness of Deep Learning Systems," arXiv:1803.07519, 2018
17. Y. Sun, X. Huang, and D. Kroening, "Testing Deep Neural Networks," arXiv:1803.04792, 2018

18. K. J. Hayhurst, D. S. Veerhusen, J. J. Chilenski, and L. K. Rierson, "A Practical Tutorial on Modified Condition/Decision Coverage," Vol. 210876, DIANE Publishing, 2001
19. Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated Testing of Deep-Neural-Network-Driven Autonomous Cars," arXiv:1708.08559, 2017
20. Y. Sun, "Concolic Testing for Deep Neural Networks," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pp. 109-119, 2018
21. X. Xie, L. Ma, F. Xu, M. Xue, H. Chen, Y. Liu, et al., "DeepHunter: A Coverage-Guided Fuzz Testing Framework for Deep Neural Networks," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 146-157, ACM, 2019
22. N. Nicole, M. Raugas, R. Jasper, and N. Hilliard, "Faster Fuzzing: Reinitialization with Deep Neural Models," arXiv:1711.02807, 2017