Coverage-guided Fuzzing for Feedforward Neural Networks

Xiaofei Xie[1] Hongxu Chen[1] Yi Li[1] Lei Ma[2] Yang Liu[1] Jianjun Zhao[2]
[1] Nanyang Technological University, Singapore [2] Kyushu University, Japan

***Abstract*—Deep neural network (DNN) has been widely applied to safety-critical scenarios such as autonomous vehicle, security surveillance, and cyber-physical control systems. Yet, the incorrect behaviors of DNNs can lead to severe accidents and tremendous losses due to hidden defects. In this paper, we present DeepHunter, a general-purpose fuzzing framework for detecting defects of DNNs. DeepHunter is inspired by traditional grey-box fuzzing and aims to increase the overall test coverage by applying adaptive heuristics according to runtime feedback. Specifically, DeepHunter provides a series of seed selection strategies, metamorphic mutation strategies, and testing criteria customized to DNN testing; all these components support multiple built-in configurations which are easy to extend. We evaluated DeepHunter on two popular datasets and the results demonstrate the effectiveness of DeepHunter in achieving coverage increase and detecting real defects. A video demonstration which showcases the main features of DeepHunter can be found at https://youtu.be/s5DfLErcgrc.**

I. Introduction

The past decade has witnessed the success of deep learning (DL) in many cutting-edge applications such as image processing, speech recognition, and gaming (*e.g.*, Go and Dota2). Similar to traditional software, software systems powered by deep neural networks (DNNs) (whereby referred to as DL software) also suffer from defects that can cause severe accidents and tremendous losses as happened in the Tesla/Uber crash and the malicious attacks against intelligent audiobots (*e.g.*, Siri and Alexa) through hidden commands. These incidents highlight the necessity of quality and security assurance for DL software, which should be ensured with systematic testing efforts before the software being deployed in production. Yet, testing techniques for traditional software cannot be directly applied for DL software due to two major challenges.

The first challenge lies in the *absence of test oracle* for DL software. Traditional fuzzing techniques usually rely on abnormal runtime behaviors, such as program crashes and system signals, to identify potential software defects. However, defects in DL software do not typically manifest themselves in any perceivable manner and only silently lead to incorrect decision results. Therefore, current testing efforts for DL software have to rely on pre-labeled test cases where the correct decision results are provided by a human. Such test cases are often expensive to obtain and too limited for defect detection.

The second challenge is rooted in the *lack of effective testing criteria*. The testing criterion is often used to measure the adequacy of testing efforts, which is a fundamental component in the coverage-guided test generation techniques [1]. The testing criteria for traditional software, such as line and branch coverage, are defined in terms of the program structure, however cannot be easily applied for DL software.

In this paper, we present a general-purpose coverage-guided fuzzing framework for DNNs, DeepHunter, to address these challenges and perform effective testing of DL software. The inputs of DeepHunter include a set of initial seeds and a target DNN. DeepHunter generates multiple passed tests which have high coverage and the failed ones which trigger erroneous behaviors of the DNN. DeepHunter consists of several highly configurable components, including the *seed selection*, *metamorphic mutation*, and *test coverage criteria*, which can be customized for specific application domains. To this end, DeepHunter has integrated four seed selection strategies and five recently proposed testing criteria, including Neuron Coverage (NC) [2], k-multisection Neuron Coverage (KMNC) and Neuron Boundary Coverage (NBC) [3]. DeepHunter can be easily extended to include new mutation operators, coverage criteria, as well as seed selection strategies.

We see applications of DeepHunter in several scenarios.

1) **Error Detection.** Given initial seed inputs that are correctly predicted by the DNN, DeepHunter generates tests which are slightly changed from the initial seeds but predicted incorrectly. These tests reveal the weaknesses of the DNN, which can help AI developers identify the problems and enhance the robustness of the model.
2) **Regression Testing.** Apart from failure-inducing tests, DeepHunter can also be used to generate tests that are correctly predicted and achieve high coverage in terms of a specific testing criterion. Such tests with high coverage are expected to capture more diverse behaviors of the DNN. When the model is updated, the passed tests can be used to detect errors introduced from regression.
3) **Differential Testing.** During platform migration (*e.g.*, migrating a server-side DNN to mobile devices), certain optimizations or quantizations [4] are usually performed for DL software. In such a scenario, it is necessary to detect the potential disagreements between two different models. With the fine-grained coverage guidance, DeepHunter can generate tests which manifest minor differences.

II. The DeepHunter Tool

Fig. 1 shows the overall workflow of DeepHunter. It feeds the target DL software with some initial seeds, and outputs two groups of tests, namely the failing tests that are failed by the test oracles and other passing ones stored in the seed queue.

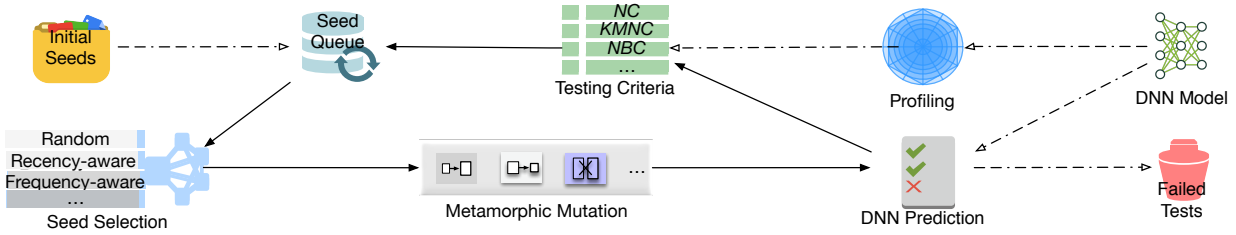

Fig. 1. The overall workflow of DeepHunter

The core of DeepHunter is a fuzzing loop which can be broken down into the seed selection, metamorphic mutation, DNN runtime prediction, and testing criteria stages. The behaviors of neurons of a given DNN model are first analyzed at the profiling stage based on the training data [3]. The results of this analysis are used to compute the coverage metrics subsequently. Seed selection (§II-A) determines the seed to be picked for mutation in the next iteration based on some predefined strategies. Metamorphic mutation (§II-B) allows different categories of mutation operators to be applied on the selected seeds. DNN prediction (§II-D) evaluates whether a newly generated seed fails the test oracles. If so, the failing seed is output as a *failed test*. Otherwise, the passing seed is further processed. The testing criteria component (§II-C) determine whether the passing seed contributes to a coverage increase, and then either appended it to the seed queue or discard it. Now we describe each stage in details.

A. Seed Selection

Seed selection preferentially picks the seeds that have high chances to be mutated to induce new coverage. DeepHunter provides three alternative built-in seed selection strategies: (1) *recency-aware strategy* which is also used by traditional coverage-guided fuzzers where seeds in the queue that induce new coverage are selected, mostly because a seed mutant is more likely to cover sibling branches of the seed which it originates from. DeepHunter integrates two recency-aware strategies from *DeepTest* and *TensorFuzz*. (2) *uniform strategy* which randomly selects a seed from the queue based on a uniform distribution. (3) *frequency-aware strategy*, which probabilistically selects a seed based on the number of times it has been fuzzed. The last strategy takes into account the diversity of mutants: if a seed has already been picked many times, it has a lower probability of being selected.

B. Metamorphic Mutation

The seed mutation component generates new seeds from the chosen seeds. Traditional fuzzers such as AFL [1] typically mutate seeds with several "primitive mutators" such as bitwise/bytewise flips, block replacement, and the crossover between input files, *etc*. The purpose is to increase the diversity of the generated seeds in the hope to cover more explicit program structures. However, these mutation operators are not effective for DNN testing, since they do not consider semantics of the input domains. For example, images generated by random mutations are often not recognizable by human beings.

Targeting at the image domain, DeepHunter implements a set of *mutators* which slightly change the seeds such that there are *no observable semantic differences* between the original seed x and its mutant x'. In other words, a *metamorphic mutation* M creates an oracle O which satisfies $O(x') = O(x)$, for $x' \in M(x)$. Given a DNN F, x' exhibits an *erroneous behavior* (or *error* for short) when $F(x) = O(x)$ while $F(x') \neq F(x)$.

Specific to DNN used for image processing, we have built-in support for eight image transformations under two categories.

- *Pixel value transformation*: image contrast, brightness, blur, and noise;
- *Affine transformation*: image translation, scaling, shearing, and rotation.

Intuitively, the former mutates pixel values, while the latter applies linear transformations on images. To constrain the difference between a seed and its mutant, DeepHunter uses a conservative strategy that selects affine transformation at most once while pixel value transformations are allowed multiple times. This is based on our experience that multiple iterations of affine transformations are prone to generate invalid images.

DeepHunter allows integration of other mutators; therefore, developers can easily add more mutators for image processing, or the counterparts in other DL software for speech recognition, natural language processing, etc.

C. Testing Criteria

At this stage, different criteria are computed to determine whether the seed mutant brings new coverage. DeepHunter has integrated five neuron-based testing criteria [2], [3]: Neuron Coverage (NC), k-multisection Neuron Coverage (KMNC), Neuron Boundary Coverage (NBC), Strong Neuron Activation Coverage (SNAC) and Top-k Neuron Coverage (TKNC). The criteria are based on the behaviors of neurons and are defined at different granularity. For example, NC measures the neurons that are activated within a predefined threshold. KMNC generalizes NC and splits the neuron output to k intervals. Different from KMNC and NC which consider the major functional range of neuron, NBC and SNAC measure how the corner-case regions can be covered. TKNC considers the neuron output from the perspective of the layer. The diverse testing criteria facilitate the test generation towards covering diversified behaviors of DNN. In the future, we plan to add more testing criteria to allow for more effective coverage guidance.

D. DNN Prediction

At this stage, it determines whether the newly generated seeds fail the test oracle given a specific DNN model. Dee-

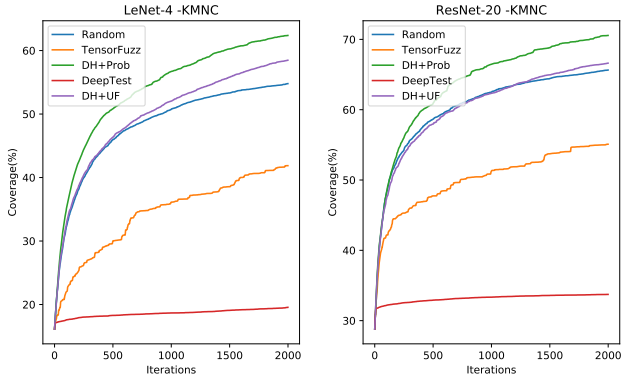

Fig. 2. Coverage increasing on LeNet-4 and ResNet-20 with KMNC guidance

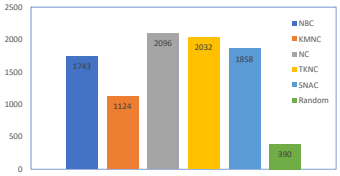

Fig. 3. Number of errors detected with different guidance on LeNet-4

pHunter compares the prediction of the original seed and its mutant: the mutant is considered to trigger an error if the two prediction results are different.

To compute the coverage of a mutant, DeepHunter maintains an array to record the covered sections for each testing criterion. For example, for NC, the size of the array is equal to the number of neurons in the DNN. The value at an index is either 0 or 1, representing whether a neuron is activated or not. For KMNC, the size of the array is $K \times \#N$, where K is the predefined number of sections and $\#N$ represents the number of neurons. The value at an index represents whether the section of a neuron is covered. Compared with the existing seeds in the seed queue, if some new sections are covered, then the coverage is increased. Currently, DeepHunter mainly supports feed-forward neural networks (FNNs). In FNNs, the values of the coverage array are either 0 or 1, because each neuron only produces one output during the prediction. For recurrent neural networks (RNNs), each neuron may produce multiple outputs because RNN has feedback loops.

E. Command Line Usage

For a target DNN, it should be profiled before any of the testing criteria from *DeepGuage* [3] is selected (*i.e.*, KMNC, NBC, SNAC or TKNC). DeepHunter provides a separate profiling procedure. Given a model, the following command generates a profiling file from the training data:

```
python profile.py -model model -train train_data
       -o profile_file
```

where `-model` specifies the path of the target model (*i.e.*, *model*), `-train` specifies the path of the training data (*i.e.*, *train_data*), and `-o` specifies the profiling output.

After the profiling stage, users can run with the following command to proceed with the fuzzing process:

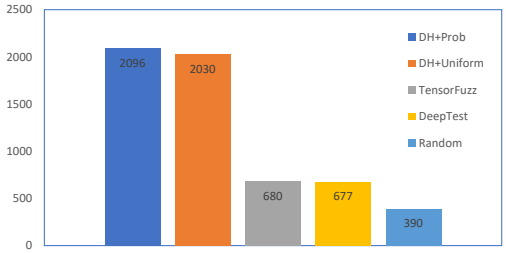

Fig. 4. Number of errors detected with different selection strategies

```
python image_fuzzer.py -i seed_dir -o outdir
       -model model -profile profile_file
       -criteria criteria_name
       -random israndom -select strategy_name
       -max_iteration iterations
```

where `-i` specifies the initial seed directory (*i.e.*, *seed_dir*), `-o` configures the path of the output directory (*i.e.*, *outdir*), `-model` sets up the DNN model to be tested (*i.e.*, *model*), `-profile` specifies the profiling file for the model, `-criteria` configures which testing criterion to be selected for coverage guidance (*i.e.*, *criteria_name*), `-random` configures whether coverage guidance is used (*i.e.*, `israndom` can be 1 or 0), `-select` specifies the seed selection strategy, and `-max_iteration` limits the maximum iterations DeepHunter executes. Note that inside the output directory, there are two subdirectories: *outdir/queue* contains tests in the seed queue and *output/crashes* contains the failed tests which do not pass the test oracle. In addition, if `israndom` is 1, then DeepHunter runs in a non-coverage guidance mode.

III. Experiments

We have implemented DeepHunter[1] as a self-contained fuzz testing framework in Python.

We demonstrate DeepHunter on two DNN models (*i.e.*, LeNet-4 and ResNet-20), which were trained from the MNIST [5] and CIFAR-10 [6] datasets, respectively. More evaluation results could be found in [7] MNIST and CIFAR-10 contain 60,000/50,000 training data and 10,000/10,000 test data. For each dataset, we randomly selected 500 test data, which can be correctly predicted by the model, as the initial seeds. We ran DeepHunter with different configurations, *i.e.*, random testing without coverage guidance, four selection strategies (*i.e.*, one frequency-aware strategy, one uniform strategy, two recency-aware strategies from *DeepTest* [8] and *TensorFuzz* [9]) and five testing criteria to test the models. Each model was tested 2,000 iterations. The experiments were conducted on a server running Ubuntu 16.04 with 28-core 2.0GHz Xeon CPU and 94 GB RAM.

Due to the space limit, we only show parts of the results. Fig. 2 shows the KMNC coverage increases with different seed selection strategies on the two models. The vertical and horizontal axes represent the coverage and the iterations finished, respectively. Compared with other seed selection strategies in

[1]The source code of DeepHunter is available at https://bitbucket.org/xiaofeixie/deephunter.

TensorFuzz [9] and *DeepTest* [8], our proposed *frequency-aware* and *uniform strategy* perform much better. Surprisingly, *random selection* also outperforms both *TensorFuzz* and *DeepTest* with the KMNC criterion. The results indicate that DeepHunter is effective in achieving improved coverage.

Fig. 3 shows the number of errors generated by DeepHunter with the *frequency-aware strategy* and different *coverage guidance strategies* on LeNet-4. The fuzzer runs for 2,000 iterations and the average time taken of each iteration is about 0.24s. The results show that: 1) DeepHunter is effective in generating error-inducing tests with different coverage guidance strategies; 2) coverage-guided testing for DL software is generally more effective than random testing (without coverage guidance); 3) Neuron Coverage (NC) tends to perform better than the others in detecting errors.

Fig. 4 depicts the number of errors detected by DeepHunter with different *seed selection strategies* and *neuron coverage guidance*. *TensorFuzz* and *DeepTest* adopt the *recency-aware selection* (*i.e*. selecting the newly generated seeds). The results show that the frequency-aware selection strategy (*i.e*. DH+Prob) and the *uniform* selection strategy (*i.e*. DH+Uniform) are more effective than both *DeepTest* and *TensorFuzz* in capturing erroneous behaviors of the DNN models.

IV. Related Work

Many issues during DL development and deployment are studied in [10]. DL software testing has recently been an active area of research covering *testing criteria* [2], [3], [11], [12] and *automated test generation* [2], [13], [8], [14], [9].

DeepXplore [2] first introduced Neuron Coverage (NC) which measures the parentage of activated neurons. *DeepGauge* [3] proposed a set of improved neuron-based metrics with different granularity. *DeepMutation* [11] is a mutation testing technique which evaluates the quality of the test data, based on source-level and model-level mutation operators. Kim *et al*. [12] proposed a set of surprise adequacy criteria based on the behaviors of DNN with respect to the training data. Surprise adequacy criteria measure how surprising an input is against a set of provided data. *DeepStellar* [15] proposed a set of metrics for recurrent neuron networks (RNNs). We also plan to integrate these criteria in the future.

Based on Neuron Coverage, *DeepXplore* [2] detects the disagreements between various DNNs with a white-box differential testing technique. *DiffChaser* [13] adopted a black-box solution which uses the genetic algorithm to search for disagreements among multiple DNNs. *DeepTest* introduced a set of mutation operators which are used to generate tests for testing DNNs with guidance from Neuron Coverage. *DeepConclic* [14] implemented a concolic testing technique for DNNs and it generates tests with the help from constraint solving. Similar to DeepHunter, *TensorFuzz* is a coverage-guided fuzzing technique that uses the approximate nearest neighbor to guide the test generation. Compared with *TensorFuzz*, DeepHunter provides mutation operators, fine-grained neuron-based testing criteria and the frequency-aware seed selection strategy.

V. Conclusion

In this paper, we described the architecture of DeepHunter including its metamorphic mutation, coverage guidance, and seed selection strategies. We demonstrated the effectiveness of our tool in testing DL software with two DNN models from two public datasets. DeepHunter is highly extensible to process different types of models, handle diverse application domains, select and mutate seeds with various strategies.

Acknowledgments

This research was supported (in part) by the National Research Foundation, Prime Ministers Office Singapore under its National Cybersecurity R&D Program (Award No. NRF2018NCR-NCR005-0001), National Satellite of Excellence in Trustworthy Software System (Award No. NRF2018NCR-NSOE003-0001) administered by the National Cybersecurity R&D Directorate, the Singapore Ministry of Education Academic Research Fund Tier 1 (No. 2018-T1-002-069) and JSPS KAKENHI Grant 19K24348, 19H04086, and Qdaijump Research Program NO.01277. We also thank Nvidia for the hardware support including GPU cards and servers, and Mr. Yun Tang for the great help when preparing the demo.

References

[1] " American Fuzzy Lop ," 2018. [Online]. Available: http://lcamtuf.coredump.cx/afl/

[2] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *SOSP*, 2017, pp. 1–18.

[3] 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 *ASE*. ACM, 2018, pp. 120–131.

[4] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[5] Y. LeCun and C. Cortes, "The MNIST database of handwritten digits," 1998.

[6] N. Krizhevsky, H. Vinod, C. Geoffrey, M. Papadakis, and A. Ventresque, "The cifar-10 dataset," http://www.cs.toronto.edu/kriz/cifar.html, 2014.

[7] X. Xie, L. Ma, F. Juefei-Xu, M. Xue, H. Chen, Y. Liu, J. Zhao, B. Li, J. Yin, and S. See, "Deephunter: a coverage-guided fuzz testing framework for deep neural networks," in *ISSTA*. ACM, 2019, pp. 146–157.

[8] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *ICSE*. ACM, 2018, pp. 303–314.

[9] A. Odena and I. Goodfellow, "Tensorfuzz: Debugging neural networks with coverage-guided fuzzing," in *ICML*, 2019.

[10] Q. Guo, S. Chen, X. Xie, L. Ma, Q. Hu, H. Liu, Y. Liu, J. Zhao, and X. Li, "An empirical study towards characterizing deeplearning development and deployment acrossdifferent frameworks and platforms," in *ASE*. ACM, 2019.

[11] 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," *ISSRE*, 2018.

[12] J. Kim, R. Feldt, and S. Yoo, "Guiding deep learning system testing using surprise adequacy," in *Proceedings of the 41st International Conference on Software Engineering*, ser. ICSE '19, 2019, pp. 1039–1049.

[13] X. Xie, L. Ma, H. Wang, Y. Li, Y. Liu, and X. Li, "Diffchaser: Detecting disagreements for deep neural networks," in *IJCAI*, 2019, pp. 5772–5778.

[14] Y. Sun, M. Wu, W. Ruan, X. Huang, M. Kwiatkowska, and D. Kroening, "Concolic testing for deep neural networks," in *ASE*, ser. ASE 2018, 2018, pp. 109–119.

[15] X. Du, X. Xie, Y. Li, L. Ma, Y. Liu, and J. Zhao, "Deepstellar: Model-based quantitative analysis of stateful deep learning systems," in *FSE*, 2019, pp. 477–487.