DeepController: Feedback-Directed Fuzzing for Deep Learning Systems

Hepeng Dai [1] Chang-ai Sun [1] * Huai Liu [2]

[1] University of Science and Technology Beijing
Email: daihepeng@sina.cn, casun@ustb.edu.cn

[2] Swinburne University of Technology
Email: hliu@swin.edu.au

Abstract

Deep learning (DL) systems are increasingly adopted in various fields, while fatal failures are still inevitable in them. One mainstream testing approach for DL is fuzzing, which can generate a large amount of semi-random yet syntactically valid test cases. Previous studies on fuzzing are mainly focused on selecting "quality" seeds or using "good" mutation strategies. In this paper, we attempt to improve the performance of fuzzing from a different perspective. A new fuzzer, namely DeepController, is accordingly developed, which makes use of the feedback information obtained in the test execution process to dynamically select seeds and mutation strategies. DeepController is evaluated through empirical studies on three datasets and eight DL models. The experimental results show that, with the same number of seeds, DeepController can generate more adversarial inputs and achieve higher neuron coverage than the state-of-the-art testing techniques for DL systems.

Keywords: *Fuzzing, Deep Learning Systems, Software Testing*

1 Introduction

Nowadays, deep learning (DL) systems are used in a wide variety of fields, thanks to their powerful learning and reasoning capabilities. Nevertheless, DL systems, like traditional software systems, unavoidably contain some faults and thus show incorrect or unexpected behaviors. Testing is a main approach to support quality assurance of software systems. However, the unique features of DL systems pose new challenges for testing. For example, the logic behind a DL system is not manifested as that in traditional software; DL introduces much higher non-determinism in the software output. As such, many traditional testing techniques are no longer applicable to DL systems.

Among recently proposed testing techniques for DL systems, fuzz testing (or simply *fuzzing*) [1] is a basic technique that has gradually become a standard method in the industry. It can generate lots of semi-random test cases based on existing test data with relatively low cost. Despite the simplicity in concept, fuzzing has successfully generated the *adversarial inputs* that facilitate the fault detection [2]. Since the first white-box-based fuzzing method for DL systems [1], various fuzzing techniques have been proposed [3]. Some fuzzing techniques are focused on selecting appropriate seeds [1, 4], while others attempt to improve the performance of fuzzing via choosing "good" mutation strategies [5, 6].

Most previous studies on seed selection mainly used single pieces of test information (e.g., coverage information, the number of seed mutations, or the times of seeds added to the seed queue), but did not consider a variety of information comprehensively. Previous studies on mutation strategies showed that they tend to activate different sets of neurons [7], implying the uncertainty on the optimal mutation strategies for a specified seed during testing. Unfortunately, existing mutation strategy selection approaches did not individually consider specific seeds, which may affect the fault-detection efficiency of fuzzing.

In this paper, we propose a new fuzzing technique, namely *DeepController*, which makes full use of the feedback information (including coverage information, testing results, the number of seed mutations, and the times of seeds added to the seed queue) collected during the test execution process to guide the selections of seeds and mutation strategies. In line with software cybernetics [8], DeepController treats the whole testing procedure as a feedback control system, providing feedback-directed strategies for selecting seeds from the seed queue and choosing the most appropriate mutation strategies for selected seeds.

Our study has the following three major contributions:

(1) A comprehensive framework (Section 3.1), was proposed to implement DeepController, which can select seeds and mutation strategies that both have high potentials of fault detection, adaptive to the feedback information from test executions.
(2) Two algorithms, namely AS2 (Section 3.2) and AMS2 (Section 3.3), were developed for the selections of seeds and mutation strategies, respectively.
(3) A series of empirical studies on three datasets, and

*corresponding author

DOI reference number: 10.18293/SEKE2022-126

eight DL models (Section 4), were conducted to evaluate the performance of DeepController. As observed from these experiments, DeepController could generate more adversarial inputs, and obtain higher neuron coverage than the state-of-the-art DL testing techniques.

2 Background and Related Work

2.1 Fuzzing

The key idea of fuzzing is to generate a large amount of semi-random yet syntactically valid test cases by mutating existing test cases. Generally, the test case generation is composed of the following three components.

- *Seed queue construction*, which is responsible for constructing the seed queue by selecting test cases from corpus that contains a test pool as well as the label and coverage information of each test case.
- *Seed selection*, which is responsible to select seeds from the seed queue based on a certain selection strategy.
- *Seed mutation*, which uses some seed mutation strategies to generate mutated seeds that serve as test cases for executing the program under test (PUT).

2.2 Related Work

Recently, researchers have proposed a large number of fuzzing techniques for DL systems based on different theories and observations. Among them, coverage-guided fuzzing techniques have been proven to be very effective in detecting faults and exploring the internal states of the DL models. Closely related works are described below.

Seed Selection. Fuzzing iteratively selects seeds from the seed queue based on some strategies. One simple and commonly used strategy is random selection, but it does not use any information of testing process or DL systems. To enhance the random strategy, the idea of "recency-aware" was used to select the seed that induces the new coverage [9, 7, 4, 10, 11, 12], which corresponds to the observation that if a seed covers a branch, the following branches are more likely to be covered due to the hierarchical relationship between branches. In addition, the idea of "frequency-aware" was used to probabilistically select a seed based on the number of times it has been mutated: if a seed has already been picked many times, it has a lower probability of being selected again [4, 10].

Seed Mutation. As one core component of fuzzing, mutation strategy directly affects the fault-detection efficiency and effectiveness of fuzzing. The existing strategies for seed mutation can be classified into three categories: (1) gradient-based mutation strategies, which first calculate the gradient of the objective function, and then mutate the seed according to the calculated gradient [1, 13, 14]; (2) domain-knowledge-based mutation strategies, which mutate the seeds according to the properties of inputs, while the mutated seed has the same semantic information as the original one [7, 15, 16]; (3) search-based mutation strategies, which mutate the selected seeds using some search-based algorithms, such as population-based metaheuristics [5] and Monte Carlo Tree Search (MCTS) [6].

Although existing fuzzers showed promising results in detecting faults in DL systems, they do not make full use of the information obtained in test execution process, which could be useful for improving the efficiency of fuzzing, thus motivating this study.

3 Methodology

In order to further improve the performance of fuzzing for DL systems, this study makes use of the feedback information obtained in test executions to guide the selection of appropriate seeds and proper mutation strategies, particularly focused on designing new framework and two strategies for seed and mutation strategy selections.

3.1 Framework

Based on the principles of software cybernetics and the features of fuzzing, we propose the framework of DeepController, as illustrated in Figure 1. The starting point of DeepController is that there already exist a test suite and some mutation strategies. Note that researchers have devised many mutation strategies for different types of DL models, so it remains a challenging issue how to select them for testing. There is a feedback loop in the framework, which consists of basic components of fuzzing, DL model, the database for storing test information (including coverage information, the testing results, the times of the seed being selected, and the time of the seed adding to seed queue), and the controller for selecting seed and mutation strategy. Particularly, in the controller, the historical test information is leveraged to guide the selections of seeds and mutation strategies. Furthermore, the historical information can also be used to improve the underlying testing strategies.

3.2 Seed Selection Strategy

Suppose that a seed queue Q has n seeds, that is, $Q = \{s_1, s_2, \ldots, s_n\}$. $L_i(i = 1, 2, \ldots, n) = \{(s_i, t_0)\}$ is to store the time of s_i and the seeds generated based on s_i added to Q. For instance, seeds s_1^* and s_2^* are generated at time t_1^* and t_2^* by seeds s_1 and s_2, respectively. Both s_1^* and s_2^* can trigger some new coverage (e.g. new neurons). Then L_1 and L_2 should be updated as $L_1 = L_1 \cup \{(s_1^*, t_1^*)\}$ and $L_2 = L_2 \cup \{(s_2^*, t_2^*)\}$. The selection probabilities of L_i are denoted as $LP = \{\langle L_1, p_1\rangle, \langle L_2, p_2\rangle, \ldots, \langle L_n, p_n\rangle\}$, where $p_i(i = 1, \ldots, n)$ denotes the selection probability of L_i. There are two lists $E = \{e_1, e_2, \ldots, e_n\}$ and $E' = \{e'_1, e'_2, \ldots, e'_n\}$, where $e_i(i = 1, 2, \ldots, n)$ records the times the seeds in L_i are selected and trigger new coverage, while e'_i records the times the seeds in L_i are selected but no new coverage is triggered.

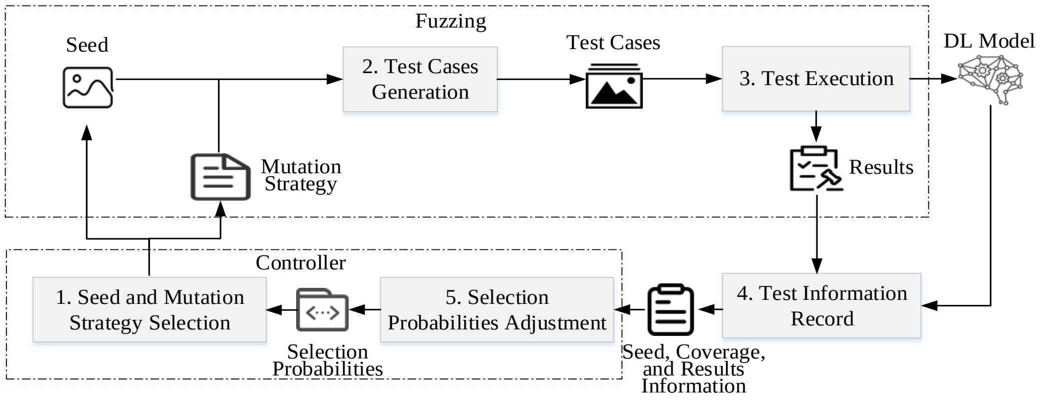
Fuzzing
Seed
2. Test Cases Generation
Test Cases
3. Test Execution
DL Model
Mutation Strategy
Results
Controller
1. Seed and Mutation Strategy Selection
5. Selection Probabilities Adjustment
4. Test Information Record
Selection Probabilities
Seed, Coverage, and Results Information

Figure 1: The framework of DeepController

We propose an adaptive seed selection strategy, namely AS2, which utilizes test information (including the coverage, the times of seeds being used, and the time of a seed added to seed queue) to select those seeds with higher fault-detection potentials. If the mutated seeds generated by s_i (that belonged to L_i) could trigger new coverage, the corresponding selection probability p_i of L_i will be increased; otherwise, p_i will be decreased. Moreover, the smaller the times of L_i being selected, the increase of p_i is greater; otherwise, the increase of p_i is smaller.

At the beginning, AS2 initializes $LP = \{\langle L_1, 1/n\rangle, \langle L_2, 1/n\rangle, \dots, \langle L_n, 1/n\rangle\}$, and $e_i = 0, e'_i = 0 (i = 1, \dots, n)$. During the test process, assume that AS2 selects L_i based on the LP, and the seed s_i that is the latest seed of L_i is selected. Accordingly, a set of mutated seeds $T = \{s_i^1, s_i^2, \dots, s_i^k\}$ (k is the number of times a seed can be mutated) are generated based on s_i and some mutation strategies. Suppose that all seeds in T are executed. If $\exists s_i* \in T$, and s_i* triggers new coverage and $\forall j = 1, \dots, n, j \neq i$, we then update $e_i = e_i + 1$ and set

$$p'_j = \begin{cases} p_j - \dfrac{\epsilon \times (1 + ln(1 + 1/(e_j + 1)))}{n-1} & \text{if } p_j \geq W \\ 0 & \text{if } p_j < W \end{cases}, \tag{1}$$

where ϵ is a probability adjusting factor, and $W = \dfrac{\epsilon \times (1 + ln(1 + 1/(e_j + 1)))}{n-1}$. Then,

$$p'_i = 1 - \sum_{j=1, j\neq i}^{n} p'_j. \tag{2}$$

Alternatively, If $\forall s_i* \in T$, and s_i* cannot trigger new coverage, we then update $e'_i = e'_i + 1$ and set

$$p'_i = \begin{cases} p_i - \epsilon \times (1 + e'_i/n) & \text{if } p_i \geq \epsilon \times (1 + e'_i/n) \\ 0 & \text{if } p_i < \epsilon \times (1 + e'_i/n) \end{cases}, \tag{3}$$

$$p'_j = \begin{cases} p_j + \dfrac{\epsilon \times (1 + e'_i/n)}{n-1} & \text{if } p_i \geq \epsilon \times (1 + e'_i/n) \\ p_j + \dfrac{p_i}{n-1} & \text{if } p_i < \epsilon \times (1 + e'_i/n) \end{cases} \tag{4}$$

.

AS2 keeps updating the selection probabilities of seeds via the formulas 1 to 4. As a result, the seeds with higher chances of triggering new coverage and being used less times have higher probabilities of being selected.

3.3 Mutation Strategy Selection Strategy

Suppose that there exist a seed queue $Q = \{s_1, s_2, \dots, s_n\}$ and a set of mutation strategies $M = \{m_1, m_2, \dots, m_x\}$. The list $MP_i (i = 1, 2, \dots, n) = \{\langle m_1, p_i^1\rangle, \langle m_2, p_i^2\rangle, \dots, \langle m_x, p_i^x\rangle\}$ can be created as the set of mutation strategy selection probabilities for each seed in Q. The list $C_i = \{c_i^1, c_i^2, \dots, c_i^x\}$ records the times each strategy $m_h (h = 1, \dots, x)$ has been used by s_i and triggered new coverage.

We propose an adaptive mutation strategy selection approach, namely AMS2, which analyzes the performance of mutation strategies on different seeds and selects the most appropriate strategy for a specific seed. If the mutated seeds generated by seed s_i and the mutation strategy $m_h (h = 1, 2, \dots, x)$ could trigger new coverage or detect faults, the corresponding selection probability p_i^h of m_h will be increased; otherwise, p_i^h will be decreased. Moreover, the smaller the times of m_h being used, the increase of p_i^h is greater; otherwise, the increase of p_i^h is smaller.

At the beginning, AMS2 initializes $MP_i = \{\langle m_1, 1/x\rangle, \langle m_2, 1/x\rangle, \dots, \langle m_x, 1/x\rangle\}$, and sets all elements in C_i to 0. During the test process, assume that s_i is selected by AS2. AMS2 selects a mutation strategy m_h based on the MP_i. Accordingly, a set of mutated seeds $T = \{s_i^1, s_i^2, \dots, s_i^k\}$ (k is the number of times a seed can be mutated) are generated based on the s_i and m_h. Suppose that all seeds in T are executed. If $\exists s_i* \in T$, and s_i* triggers new coverage or detects a fault, $\forall y = 1, 2, \ldots, x$ and $y \neq h$, we then update $c_i^h = c_i^h + 1$ and set

$$p_i^{y\prime} = \begin{cases} p_i^y - \dfrac{\delta \times x}{c_i^y + 1} & \text{if } p_i^y \geq \dfrac{\delta \times x}{c_i^y + 1} \\ 0 & \text{if } p_i^y < \dfrac{\delta \times x}{c_i^y + 1} \end{cases}, \quad (5)$$

where δ is a probability adjusting factor. Then,

$$p_i^{h\prime} = 1 - \sum_{y=1, y\neq h}^{x} p_i^{y\prime}. \quad (6)$$

Alternatively, If $\forall s_i* \in T$, and s_i* cannot trigger new coverage or detect a fault, we then set

$$p_i^{h\prime} = \begin{cases} p_i^h - \delta & \text{if } p_i^h \geq \delta \\ 0 & \text{if } p_i^h < \delta \end{cases}, \quad (7)$$

$$p_i^{y\prime} = \begin{cases} p_i^y + \dfrac{\delta}{x-1} & \text{if } p_i^h \geq \delta \\ p_i^y + \dfrac{p_i^h}{x-1} & \text{if } p_i^h < \delta \end{cases}. \quad (8)$$

AMS2 dynamically adjusts the selection probabilities of mutation strategies based on the formulas 5 to 8. As a result, the mutation strategies with a higher probabilities of triggering new coverage and detecting faults for the selected seeds have higher probabilities of being selected.

4 Empirical Study

We conducted a series of empirical studies to evaluate the performance of DeepController.

4.1 Research Questions

In our experiments, we focused on addressing the following three research questions.

RQ1 Can DeepController generate more adversarial inputs than the commonly used testing techniques for DL models?

RQ2 Can DeepController achieve higher neuron coverage (NC) [1] than state-of-the-art techniques?

RQ3 How about the performance of DeepController in terms of time overhead?

4.2 Experimental Design

DeepController was implemented as a self-contained fuzzing framework, written in Python based on the DL framework Keras (ver.2.1.6) with TensorFlow (ver.1.5.0) backend. With DeepController, we performed a comparative study to answer the three research questions raised above.

Datasets, DNN Models, and Baselines. We selected three popular publicly available datasets (i.e., MNIST [17], ImageNet [18], and CIFAR10 [19]) as the evaluation subjects (see Table 1). We further utilized the commonly used DL models, including LeNet-1, LeNet-4, LeNet-5, VGG-16 and VGG-19 for ImageNet, VGG-16 and VGG-19 for CIFAR10, and 20 layer CNN with max-pooling and dropout layers [6]. Note that most used models are open-source available, except VGG-16 and VGG-19 for CIFAR10, which were trained by ourselves. As summarized in a survey [3], there are several open-source tools for DL systems. To further measure the fault-detection ability of DeepController, we selected three representative fuzzers (DeepXplore [1], DeepTest [7], and DeepHunter [4]) and a gradient-based testing approach proposed recently (FGSM [20]) as our baselines.

Table 1: Subject datasets and DL models

Dataset	DL Model	Number of Parameters	Acc.(%)
ImageNet	VGG-16	138,357,544	92.60
	VGG-19	143,667,240	92.70
MNIST	LetNet-1	7,206	98.25
	LetNet-4	69,362	98.75
	LetNet-5	107,786	98.63
CIFAR10	VGG-16	138,357,544	86.84
	VGG-19	143,667,240	77.26
	CNN-20	952,234	77.68

Mutation Strategies. We select eight mutation strategies for image, which can be partitioned into two categories: (1) *Pixel Value Mutation* P, which includes image contrast, image brightness, image blur, and image noise; (2) *Affine Mutation* G, which includes image translation, image scaling, image shearing, and image rotation. The empirical results in [7] showed that combining different image mutation strategies, neuron coverage can be improved. In order to keep the semantics of the mutated seeds close to the original one, we adopt a conservative strategy that selects a pixel value mutation strategy p from P by using AMS2, and randomly selects an Affine mutation g from G. Then the mutated seeds could be obtained by applying selected p and g on the selected seed. Note that this study aims to improve the fault-detecting efficiency of fuzzing, the strategy proposed in [4] was used to judge whether the mutated seeds are valid or not.

Parameter Settings. The hyper-parameters of DeepXplore, DeepTest, DeepHunter, and FGSM were configured based on the settings in their original studies. For the probability adjusting factors ϵ and δ of DeepController, we conducted a series of trial experiments to find a fair setting, and finally set $\epsilon = 0.01$ and $\delta = 0.1$.

To reduce the randomness effect of experiments, we randomly generated ten seed queues for each dataset using different random seeds (each seed queue has 1000 images) and averaged the results. The termination condition of testing

Table 2: Average number of adversarial inputs generated by fuzzers on different datasets

Fuzzers	MNIST			ImageNet		CIFAR10		
	LeNet-1	LeNet-4	LeNet-5	VGG-16	VGG-19	VGG-16	VGG-19	CNN-20
DeepXplore	23.4	32.8	23.2	118.0	124.8	88.2	92.2	103.0
DeepTest	21.8	26.4	20.2	130.2	166.4	302.0	309.6	370.4
DeepHunter	24.8	24.4	38.6	132.6	150.4	381.4	308.2	472.6
FGSM	27.2	19.8	19.4	30.8	31.0	324.4	311.4	595.2
DeepController	**37.8**	**35.4**	**43.4**	**162.6**	**199.0**	**478.8**	**447.4**	**697.2**

Table 3: Average neuron coverage of fuzzers on different datasets

Fuzzers	MNIST			ImageNet		CIFAR10		
	LeNet-1	LeNet-4	LeNet-5	VGG-16	VGG-19	VGG-16	VGG-19	CNN-20
DeepXplore	44.62%	55.81%	53.88%	23.66%	22.20%	5.53%	5.37%	32.55%
DeepTest	47.88%	64.32%	55.60%	32.33%	29.25%	5.74%	5.24%	34.56%
DeepHunter	46.62%	63.86%	58.58%	26.98%	24.04%	5.47%	4.85%	34.26%
FGSM	38.46%	61.08%	57.39%	32.02%	28.95%	5.06%	4.76%	33.26%
DeepController	**48.08%**	**64.59%**	**59.11%**	**32.94%**	**29.80%**	**5.88%**	**5.57%**	**34.71%**

DL models on MNIST and CIFAR10 is the generation of 1000 test cases. Since testing the VGG models on ImageNet required huge testing resources, we set the generation of 500 test cases as the termination condition. Besides, in all experiments, we set the threshold of NC to 0.75 and the experiments are preformed on an Mac machine (one Intel i5 3733 MHz processor with four cores, 16GB of memory).

4.3 Results

RQ1: Generation of Adversarial Inputs. Table 2 reports the average number of adversarial inputs generated by different fuzzers. It is clearly shown that DeepController generated more adversarial inputs than the four baseline techniques.

We also conducted statistical testing to verify the significance of this evaluation. We used ANOVA [21] (with significance level $\alpha = 0.05$) to determine which pairs of testing techniques had significant differences. Our calculated results (the *f-ratio* value was 6.0951, and the *p-value* was 0.0001) show that DeepController was significantly better than the four baseline techniques in terms of the capabilities of generating adversarial inputs.

Answer to RQ1: DeepController generated more adversarial inputs than the four baseline techniques .

RQ2: Neurons Coverage. Table 3 reports the average neuron coverage achieved by the four baseline techniques and DeepController on different datasets. It is clearly shown that DeepController achieved higher neuron coverage on different models than the four baseline techniques.

Answer to RQ2: DeepController covered more neurons than the four baseline techniques.

RQ3: Time Overhead. DeepController makes use of feedback information to select appropriate seeds and mutation strategies, which might result in longer computation time as compared with DeepXplore, DeepTest, DeepHunter, and FGSM. The time overhead is mainly composed of the following: (1) *seed and mutation strategy selection time* that refers to how long it takes to select seeds and mutation strategies; (2) *seed mutation time* that refers to how long it takes to generate mutated seeds based on the selected mutation strategies and seeds; (3) *seed execution time* that represents the time required for executing mutated seeds. Table 4 reports the time overhead of the different techniques, where $x/y/z$ denotes the average time overheads of studied techniques on MNIST, ImageNet, and CIFAR10, respectively.

DeepController needed more time to select seeds. However, seed selection is an inexpensive process. On average, seed selection time was only 1.2% of the whole testing time. In terms of the whole test overhead, DeepController did not always have the longest testing time. Specifically, DeepController had shorter testing time than DeepXplore and FGSM on all datasets, shorter than DeepHunter on MNIST and CIFAR10, DeepTest had shorter testing time than DeepController on all datasets, but the difference was marginal.

Answer to RQ3: For executing the same number of test cases, DeepController generally had shorter testing time than DeepXplore, FGSM, and DeepHunter, but had marginally higher time overhead than DeepTest.

Table 4: Time overhead of the different techniques on different datasets

Techniques	Seed and Mutation Strategy Selection (s)	Seed Mutation (s)	Seed Execution (s)	Total (s)
DeepXplore	0.003/0.043/0.006	170.467/994.904 /596.079	4.658/3223.119/230.186	175.129/4218.066/826.271
DeepTest	0.003/0.015/0.053	0.384 /7.643 /0.447	1.817/1443.383/21.819	**2.204 /1451.041/22.319**
DeepHunter	0.880/0.150/0.147	157.648/31.388 /51.072	1.666/1440.198/21.035	160.194/1471.736/72.253
FGSM	0.002/0.013/0.020	4.048 /1497.753/131.868	1.681/441.829 /60.696	5.731 /1939.595/192.585
DeepController	0.782/0.087/0.151	2.204 /11.068 /1.676	1.790/1483.132/22.342	**4.776 /1494.288/24.169**

5 Conclusions and Future Work

Fuzzing has increasingly been proven to be very effective in detecting faults and exploring the internal states of the DL models. In recent years, researchers have proposed quite a few fuzzers. Nevertheless, the execution information has not been fully utilized in the state-of-the-art fuzzing techniques. In this paper, we introduced feedback into fuzzing for DL models, and proposed *DeepController*, which makes use of feedback information obtained in test executions to guide the selection of seeds and mutation strategies. Empirical studies were conducted to evaluate the performance of DeepController, in comparison with four popular techniques for testing DL models, based on three commonly used datasets, and eight DL models. The experimental results showed that the proposed approach can generate more adversarial inputs and explore more internal states of DL models, with at least similar time overhead.

For our future work, there are two aspects that need further investigations: (1) We will apply DeepController to applications with other types of inputs, such as audios and text; (2) It is also important to investigate the influence of hyper-parameters (the probability adjusting factor ϵ of AS2, and the probability adjusting factor δ of AMS2) on the performance of DeepController.

6 Acknowledgment

This work is supported by the National Natural Science Foundation of China under Grant No. 61872039 and the Fundamental Research Funds for the Central Universities under Grant No. FRF-GF-19-019B.

References

[1] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated white-box testing of deep learning systems," in *Proceedings of SOSP'17*, 2017, pp. 1–18.

[2] H. Liang, X. Pei, X. Jia, W. Shen, and J. Zhang, "Fuzzing: State of the art," *IEEE Transactions on Reliability*, vol. 67, no. 3, pp. 1199–1218, 2018.

[3] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Transactions on Software Engineering*, vol. 48, no. 1, pp. 1–36, 2020.

[4] 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 *Proceedings of ISSTA'19*, 2019, pp. 146–157.

[5] H. B. Braiek and F. Khomh, "Deepevolution: A search-based testing approach for deep neural networks," in *Proceedings of ICSME'19*, 2019, pp. 454–458.

[6] S. Demir, H. F. Eniser, and A. Sen, "Deepsmartfuzzer: Reward guided test generation for deep learning," *arXiv:1911.10621*, 2019.

[7] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of ICSE'18*, 2018, pp. 303–314.

[8] H. Yang, F. Chen, and S. Aliyu, "Modern software cybernetics: New trends," *Journal of Systems and Software*, vol. 124, pp. 169–186, 2017.

[9] A. Odena, C. Olsson, D. Andersen, and I. Goodfellow, "Tensorfuzz: Debugging neural networks with coverage-guided fuzzing," in *Proceedings of ICML'19*, 2019, pp. 4901–4911.

[10] X. Xie, H. Chen, Y. Li, L. Ma, Y. Liu, and J. Zhao, "Coverage-guided fuzzing for feedforward neural networks," in *Proceedings of ASE'19*, 2019, pp. 1162–1165.

[11] P. Zhang, B. Ren, H. Dong, and Q. Dai, "Cagfuzz:coverage-guided adversarial generative fuzzing testing for image-based deep learning systems," *IEEE Transactions on Software Engineering*, 2021. DOI: 10.1109/TSE.2021.3124006.

[12] P. Zhang, Q. Dai, and S. Ji, "Condition-guided adversarial generative testing for deep learning systems," in *Proceedings of AITest'19*, 2019, pp. 71–72.

[13] J. Guo, Y. Jiang, Y. Zhao, Q. Chen, and J. Sun, "Dlfuzz: Differential fuzzing testing of deep learning systems," in *Proceedings of ESEC/FSE'18*, 2018, pp. 739–743.

[14] S. Lee, S. Cha, D. Lee, and H. Oh, "Effective white-box testing of deep neural networks with adaptive neuron-selection strategy," in *Proceedings of ISSTA'20*, 2020, pp. 165–176.

[15] X. Du, X. Xie, Y. Li, L. Ma, J. Zhao, and Y. Liu, "Deepcruiser: Automated guided testing for stateful deep learning systems," *arXiv:1812.05339*, 2018.

[16] A. Rios, "Fuzze: Fuzzy fairness evaluation of offensive language classifiers on african-american english," in *Proceedings of AAAI'20*, vol. 34, no. 01, 2020, pp. 881–889.

[17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, vol. 25, pp. 1097–1105, 2012.

[19] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009, TR-2009.

[20] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proceedings of ICLR'15*, 2015, pp. 134–146.

[21] L. St, S. Wold *et al.*, "Analysis of variance (anova)," *Chemometrics and intelligent laboratory systems*, vol. 6, no. 4, pp. 259–272, 1989.