FTDG: Fuzzing-Based Test Data Generation for Deep Neural Networks

Ziqi Chen[1], Chuanqi Tao[1,2,3] (✉), Tianzi Zang[1], and Hongjing Guo[1]

[1] College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China
{lightt,taochuanqi,zangtianzi,guohongjing}@nuaa.edu.cn
[2] Ministry Key Laboratory for Safety-Critical Software Development and Verification, Nanjing University of Aeronautics and Astronautics, Nanjing, China
[3] Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing University, Nanjing, China

Abstract. With the widespread application of deep neural network (DNN) models, quality issues in deep learning (DL) systems have become increasingly prominent. One such issue is the insufficient test data, which fails to adequate test the decision-making behavior of DNN models. Existing research primarily focuses on fuzzing-based test data generation methods. However, these methods overlook the importance of seed queues and error-prone neurons, leading to a lack of diversity in the generated data and difficulty in revealing errors. Different from existing work, this paper proposes FTDG, a novel test data generation approach for DNN models. Firstly, FTDG designs the seed selection strategy based on model prediction uncertainty to create a seed list with diversity and high uncertainty. Secondly, we design the neuron selection strategy based on the relevance of model prediction errors to provide the necessary neurons for the mutation strategy. Finally, we iteratively mutate the seeds in the selected seed list until adversarial test data is generated. Experimental results show that FTDG outperforms the compared methods, with an average neuron coverage improvement of 5.1% to 94.1%, and an increase of at least 35.7% and 38.6% in the number and diversity of generated test data, respectively.

Keywords: Deep neural networks · Fuzz testing · Test data generation · Test coverage.

1 Introduction

With the rapid development of artificial intelligence, deep neural networks (DNNs) have been widely applied in image recognition [16], image inpainting [10], machine translation [23], autonomous driving [29],etc. However, due to the data-driven nature of DNN models and their complex internal structures, even minor perturbations in input data can cause significant deviations in model output. For example, in autonomous driving scenarios, such perturbations may lead to errors in traffic sign recognition, which could result in traffic accidents. Therefore,

to ensure the reliability and robustness of DNN systems in dynamic environments, conducting comprehensive and systematic testing has become critically important.

In DNN testing, due to the complexity and variability of real-world scenarios as well as the unpredictability of edge cases, the distribution space of data is infinite. And it is difficult to cover the distribution of data in all scenarios by manually collecting, leading to a lack of diversity and effectiveness in the test inputs (i.e., test data). Consequently, the decision-making behavior of the model cannot be thoroughly tested.

Recently, researchers have proposed a range of test data generation methods to address the aforementioned issues. A notable class of methods includes coverage-guided fuzz testing (CGF) techniques, such as DeepXplore [24], DLFuzz [8],ADAPT [15]. However, existing research primarily uses random methods to select a certain number of seeds from the test set to construct a seed list. This approach does not effectively utilize the output information of test data on DNN models and fails to filter out diverse seeds with high output uncertainty to guide the generation of test data. Meanwhile, existing neuron selection strategies are mainly based on heuristics or assumptions, lacking a thorough understanding of the selected neurons and the underlying reasons for their selection. This makes it difficult to assess the contribution of the selected neurons to coverage improvement, ultimately resulting in poor performance of test data generation methods.

To address the above issues, we propose FTDG, a novel fuzz testing framework for test data generation. In order to construct a seed queue with high uncertainty and diversity for generating error-revealing and diverse test data, FTDG designs an uncertainty-based seed queue construction strategy for the classification and regression tasks by measuring the output uncertainty of the test data on the DNN model under tested. Additionally, to address the lack of interpretability in existing neuron selection strategies, FTDG designs a relevance-based neuron selection strategy. By leveraging the Layer-wise Relevance Propagation (LRP) algorithm [1], this strategy interprets the relevance between neurons and the erroneous decision logic of DNNs, enabling the selection of neurons most related to model errors. During the mutation process to generate adversarial test data, these selected neurons are more likely to be covered and activate more neurons when triggering DNN errors, further improving the effectiveness of the CGF technique. In summary, the main contributions of this paper are:

- We design a seed selection strategy based on model prediction uncertainty, constructing seed lists with high uncertainty and diversity to enhance the effectiveness of fuzz testing.
- We design a highly interpretable neuron selection strategy based on the correlation of model prediction errors, aimed at selecting neurons related to model errors to guide the mutation of seed data.
- Extensive experiments on various classification and regression models demonstrate that FTDG surpasses existing state-of-the-art methods in both quantity and diversity, achieving higher neuron coverage.

2 Related Work

Current research on test data generation for deep neural networks can be categorized into two types: coverage-based test data generation and adversarial-based test data generation.

2.1 Coverage-based Test Data Generation

The primary approach for coverage-based test data generation is fuzz testing [19], with mutation strategies [11] serving as its core. Existing seed mutation strategies for deep learning fuzz testing are primarily categorized into gradient-based mutation methods, domain knowledge-based mutation methods, and neural network-based mutation methods, and so on.

Gradient-based mutation methods take the test objective as the objective function, calculate its derivative as the gradient, and then mutate the original seed based on this gradient. For example, Pei et al. proposed DeepXplore [24], which uses a joint objective function that maximizes neuron coverage and the variance of model outputs to generate new test data. Similarly, DLFuzz proposed by Guo et al. [8] formulates a joint objective function to maximize neuron coverage and the number of incorrect model predictions, and then uses this objective to mutate the seed inputs. ADAPT [15] adaptively selects neurons based on coverage metrics, datasets, and models to optimize the neuron selection strategy. Domain knowledge-based mutation methods mutate the seed data based on domain features without changing its semantics. For example, in the image domain, common transformation strategies include changing brightness, panning, zooming, rotating, etc., as seen in approaches like DeepTest [27], DeepHunter [30]. In the text domain, strategies such as synonym replacement and keyword insertion are adopted [4, 17]. Neural network-based mutation methods leverage Generative Adversarial Networks (GANs) [33] to generate data similar to the original seed. For example, Zhang et al. [31, 32] train a GAN and then use the trained adversarial network to generate mutated seeds from the original inputs.

In addition, some researchers adopt methods such as symbolic execution and combinatorial testing to generate test data. For example, Sun [25] and Gopinath [7] draw inspiration from the traditional software testing technique of Concolic testing [18]. They combine concrete execution and symbolic execution of DNN models based on heuristic logic to generate effective test data.

2.2 Adversarial-based Test Data Generation

Adversarial examples [26] are special samples that have been modified by intentionally added perturbations. These samples are designed with subtle changes that are imperceptible to the human eye, but they deceive the model into making incorrect predictions. Adversarial-based test data generation techniques have become a research hotspot in recent years [9], leading to a large number of related research outcomes.

Szegedy et al. [26] first introduced the concept of adversarial examples and proposed the adversarial example generation method L-BFGS. This method formulates an optimization problem with a loss function, where small perturbations are applied to an input image that is originally correctly classified, leading to its misclassification. Goodfellow et al. [6] argued that adversarial examples are caused by the linear behavior of neural networks in high-dimensional spaces and proposed the Fast Gradient Sign Method (FGSM) based on this assumption. However, since FGSM employs a single-step approach to generate adversarial examples, it may result in excessive perturbations. To address this limitation, Kurakin et al. proposed the Basic Iterative Method (BIM) [13], which uses a multi-step iterative process. Moosavi-Dezfooli et al. [20] proposed DeepFool to find the closest distance from the original input to the decision boundary of adversarial examples. Additionally, Papernot et al. introduced a targeted adversarial example generation method called the Jacobian-based Saliency Map Approach (JSMA) [22], while Nicholas Carlini and David Wagner proposed the renowned C&W attack [2].

3 Approach

3.1 Overview

Fig. 1 illustrates the overall process of FTDG. FTDG utilizes a seed queue construction strategy based on model prediction uncertainty to select a subset of tests from the test set to construct a seed queue. Then, it guides the mutation strategy process through a neuron selection strategy based on model prediction error correlation. By slightly perturbing the seed inputs in the seed list, intermediate mutated test data is generated. For classification tasks, when the original seed inputs and the intermediate mutated test data are classified into different categories by the tested DNN model, it indicates that the intermediate mutated input has caused the model to predict incorrectly, identifying it as adversarial test data; otherwise, the mutation strategy continues to perturb the intermediate mutated test data until adversarial test data is generated. For regression tasks, when the difference between the predicted values of the original seed inputs and the intermediate mutated test data exceeds the set constraints on the tested DNN model, we consider that the intermediate mutated input triggers the errors of the DNN model, classifying it as adversarial test data; otherwise, continue to perturb the intermediate mutated input using the mutation strategy. Through this process, adversarial inputs are generated to effectively reveal the errors of DNN models.

In this section, we describe FTDG from the following three aspects: (1) the uncertainty-based seed queue construction strategy; (2) Neuron selection strategy based on the correlation of model prediction errors; (3) coverage-guided fuzz testing.

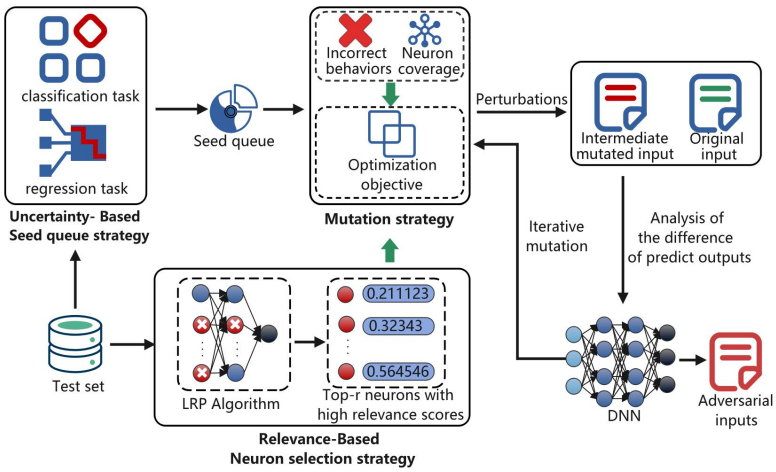


Fig. 1: The workflow of FTDG

3.2 Seed Selection Strategy Based on Model Prediction Uncertainty

The quality of seed queues is the key to ensuring the efficiency of fuzz testing. Seed queues have the following two key characteristics [3]: (1) A diverse seed queue can explore more decision logics of DNNs after mutation. (2) A seed queue with high uncertainty can detect DNNs' errors more quickly after mutation and thus improve the efficiency of fuzz testing. Based on the above findings, and considering the different task objectives and output characteristics of regression and classification tasks, FTDG designs seed selection strategies based on model prediction uncertainty for these two types of tasks. This approach aims to construct a seed list with high diversity and uncertainty, thereby improving the model's test coverage and error detection capability.

Seed Selection Strategy for Classification Tasks. DeepGini [5] is an efficient test prioritization method. This method uses the Gini coefficient to measure the uncertainty of classification confidence for a given test input on a DNN model and sorts these test inputs based on their level of uncertainty. The calculation is as follows:

$$uncertainty(t) = 1 - \sum_{i=1}^{n} p_{t,i}^2 \tag{1}$$

where $p_{t,i}$ refers the output confidence of test input t for category i. When the classification confidences of test input t from the model under test are closer to each other, it indicates that the model's output for t is more uncertain, and t is more likely to be misclassified.

To construct a seed queue with high diversity and high uncertainty, FTDG firstly divides the seeds into different categories based on their labels on the DNN model under test to ensure seed diversity; then to ensure the uncertainty of the seed queue, we use DeepGini to rank the test inputs within each category set

in descending order of uncertainty. Finally, FTDG selects the top-ranked test inputs with high uncertainty from each set to construct the seed queue. This approach ensures that the selected seeds cover different categories with high uncertainty.

Seed Selection Strategy for Regression Tasks. Dropout variance is an effective way to estimate the prediction uncertainty. Generally, the large variance of the predicted values of a test input indicates the great range of fluctuations of these output values, and thus the prediction of the test input on the DNN model under test is uncertain [12]. Therefore, FTDG uses the dropout layers to obtain the output distribution of each test input on the DNN model under test to measure the uncertainty of test inputs.

Firstly, to ensure the diversity of the seed queue, FTDG set the division distance to d and the number of predictions to k. For each test input t, FTDG performs k predictions on the model under test, obtaining k predicted values $y_{t,k}(1, ..., k)$, and calculates the average value $y_{t,avg}$. Let $y_{avg}^{\max}$ be the maximum predicted average value of all test inputs. Partition $[0, \quad y_{avg}^{\max}]$ into $y_{avg}^{\max}/d$ sets $T(0, d], T(d, 2d], ..., T(y_{avg}^{\max} - d, y_{avg}^{\max}]$. Then each test data t is added to the corresponding set according to its output average $y_{t,avg}$, for example $d < y_{t,avg} < 2d$, then t is added to $T(d, 2d]$. To ensure the uncertainty of the seed queue, the variance ν_t is calculated using $y_{t,k}$ and $y_{t,avg}$ for each t. Then ν_t is used to measure the uncertainty of the test input. For the input data in each set T, the test inputs are sorted in descending order based on ν_t. Finally, the top-ranked test inputs with high uncertainty are evenly selected from the sorted results to construct the seed queue.

3.3 Neuron Selection Strategy Based on the Relevance of Model Prediction Errors

Due to the lack of interpretability in existing neuron selection strategies, FTDG designs the neuron selection strategy based on the relevance of model prediction errors using Layer-wise Relevance Propagation (LRP) [1], an interpretable deep learning algorithm.

LRP is a deep learning model interpretation method based on deep Taylor decomposition, which utilizes the relevance scores of the input features to interpret the neural network predictions, indicating the contribution of each input feature to the model prediction results. Specifically, LRP leverages the hierarchical structure of neural networks. Based on the weights, activation functions, and input data of the model, it backpropagates the model prediction results layer by layer to the input layer. In this process, it calculates the "relevance score" for each neuron, indicating the degree of influence of the neuron on the model output. This clearly explains which input features play a key role in forming the model prediction results.

We believe that in the process of perturbing the seeds to make them predict incorrectly, it is more likely to activate neurons that provide more contribution to the incorrect prediction, namely, those neurons that have high relevance to the test inputs causing the model error (referred to as error neurons in this

paper). Therefore, FTDG selects test inputs from the test set that are incorrectly predicted by the DNN model, inputs them into the DNN model, and uses LRP to obtain relevance scores for all neurons in each layer (hidden layer) regarding the model's decision errors through the backpropagation process. These relevance scores are used to measure the actual contribution of each neuron to the incorrect predictions of these test inputs on the DNN model.

FTDG sets the number of neurons with top relevance scores to r, and selects the top r neurons with the highest relevance scores from all neurons in the hidden layers to construct the error neuron list $top_r - list$. In the process of mutation, the target neurons are always chosen from the $top_r - list$ to guide the direction of seed mutations, thereby accelerating the increase of neuron coverage.

3.4 Coverage-Guided Fuzz Testing

Coverage-guided fuzz testing mainly consists of two parts: the mutation strategy and the fuzzing strategy.

Mutation Strategy. The mutation strategy is the core of CGF which directly affects the efficiency of fault revelation on the fuzz testing. In the mutation process, FTDG iteratively mutates the original seeds of the seed queue by the seed mutation strategy to generate a large number of mutated seeds.

Due to the obvious advantages of gradient-based adversarial deep learning over other methods, FTDG leverages the existing gradient-based mutation method [8]. The core idea of this method is to model the objective with the joint optimization goal of maximizing the neuron coverage and the number of incorrect behaviors, calculate the derivatives of the objective function as gradients, and mutate the seeds using gradients.

Optimization objective function construction for classification tasks:

$$obj = \sum_{i=0}^{q} c_i - c + \lambda \sum_{i=0}^{m} n_i \tag{2}$$

FTDG defines Equation (2) as the objective function for classification tasks. The objective consists of two parts. The first part $\sum_{i=0}^{q} c_i - c$ measures the uncertainty of the test data by calculating the difference between the output probability under different labels and the output probability under the true label. Here, c is the original class label of the test data, and $c_i (i = 0, ..., q)$ is one of the top q labels other than the original label c. Maximizing the first part can guide the test data to cross the decision boundary of the original class and trigger into the decision space of the top q other classes. This modified data is more likely to be misclassified. In the second part $\sum_{i=0}^{m} n_i$, m is the number of target neurons planned to be activated, and n_i is the i-th target neuron.

Optimization objective function construction for regression tasks:

$$obj = \frac{1}{k} \sum_{i=0}^{k} \left(y_i - avg \left(\sum_{i=0}^{k} y_i \right) \right)^2 + \lambda \sum_{i=0}^{m} n_i \tag{3}$$

FTDG defines Equation (3) as the objective function for regression tasks. The objective consists of two parts. The first part $1/k \sum_{i=0}^{k} \left(y_i - a\nu g(\sum_{i=0}^{k} y_i)\right)^2$ measures the uncertainty of a test input by calculating the output variance. k is the number of predictions of the test input on the DNN model under test, y_i is the output value at the i-th prediction, and the $a\nu g(\sum_{i=0}^{k} y_i)$ is the average of k output values of the test input. Maximizing the first part can guide the test input to be mutated into the input with the highest output uncertainty, making DNN models more likely to predict incorrectly. In the second part $\sum_{i=0}^{m} n_i$, m is the number of target neurons intended to be activated, and n_i is the i-th target neuron.

The target neurons selected for both classification and regression tasks are randomly chosen from the $top_r - list$, which is constructed using the relevance-based neuron selection strategy based on model prediction errors, to improve error neuron coverage. In the mutation strategy, we use Neuron Coverage (NC) [24] as the test coverage criterion to guide the test data generation. The hyperparameter λ is used to balance the two objectives in the joint optimization goal.

Fuzzing Strategy. The fuzzing process on classification tasks is similar to it on regression tasks. Here we mainly describe the fuzzing process on regression tasks. Algorithm 1 presents the overall workflow.

Algorithm 1 Test data generation on regression tasks

Input: tested model D, test set T, seed list size n, division distance d, number of test input predictions k, proportion of erroneous neuron lists r, hyperparameter λ, number of target neurons m, iteration times $iter_{times}$.

Output: set of adversarial inputs $AdversarialSet$, neuron coverage nc

$SeedQueue \leftarrow \text{Uncertainty}(D, T, n, d, k)$
$top\_r\_list \leftarrow \text{NeuronRelevance}(D, T, r)$
for x ***in*** $SeedQueue$ **do**
  $SeedList \leftarrow [x]$ `// seeds for each input`
  while $Len(SeedList) > 0$ **do**
    $x_i \leftarrow SeedList.pop()$ `// grab the head element`
    $neuron \leftarrow \text{RandomSelection}(top\_r\_list, m)$
    $obj \leftarrow \text{Variance}(dnn.predict(x_i, k)) + \lambda \cdot \sum neuron$
    $grads \leftarrow \frac{\partial obj}{\partial x}$ `// gradient obtained`
    for $iter = 0$ ***to*** $iter_{times}$ **do**
      $x' = x^i + \text{processing}(grads)$
      update nc
      $l2_{distance} \leftarrow \text{distance}(x', x)$
      if $ImproveNC(x')$ ***and*** $Constraint(l2_{distance})$ **then**
        $SeedList$.append(x')
      if $Inconsistency(dnn, x', x)$ **then**
        $AdversarialSet$.append(x')
        Break
return $AdversarialSet, nc$

Firstly, FTDG constructs a seed queue of length using the uncertainty-based seed queue construction strategy (line 1). And then FTDG selects neurons using the relevance-based neuron selection strategy (line 2). A seed is sequentially selected from the seed queue. FTDG maintains a seed list for original seed to keep the intermediate mutated seeds that contribute to neuron coverage (lines 3-4). In the mutation process, FTDG traverses each seed of the seed list, and then randomly chooses neurons to be activated from the (lines 5-7). The objective formula is built with the joint optimization objective of maximizing the output variance of and the sum of output values of selected neurons. The gradient is calculated (lines 8-9). FTDG iteratively uses the processed gradient to perturb to obtain the intermediate mutated input (lines 10-11). After each mutation, the neuron coverage is updated, and the distance between and is calculated (lines 12-13). If the neuron coverage increases and the distance satisfies the constraint, is added to the seed list, and then the iterative mutation of continues (lines 14-15). Finally, if it causes the model to make incorrect predictions, the intermediate mutated input will be added to the set of adversarial inputs, and the mutation on is terminated (lines 16-18). Through the fuzzing process, FTDG can generate multiple adversarial inputs to reveal errors while improving the neuron coverage of DNNs (line 19).

4 Experiments

4.1 Datasets and DNN Models

For classification tasks, we adopt two popular datasets, MNIST [14] and SVHN [21], and apply two classic DNN models to each dataset. For regression tasks, we use the Udacity self-driving car image dataset [28] and apply a DNN model to this dataset. These models with different architectures are described in Table 1.

Table 1: Details of the DNNs and datasets used to evaluate FTDG.

Dataset	DNN Name	Architecture	Layer	Neuron	Accuracy/MSE
MNIST	MNI-1	LeNet-1	3	52	98.56%
	MNI-2	Custom	7	1300	99.21%
SVHN	SVH-1	ALL-CNN-A	7	2248	94.31%
	SVH-2	ALL-CNN-B	9	2824	95.21%
Driving	DAVE-drop	Dave-dropout	7	1446	0.0205

4.2 Experimental Settings

Compared Approaches. In this study, we compare FTDG with DeepXplore [24], DLFuzz [8], ADAPT [15] and Random. For Random method, its fuzzing process

is set the same as FTDG, but its seed and neuron selection strategy can be randomly chosen according to the experimental needs.

FTDG, DeepXplore, and Random can be used for classification and regression tasks, ADAPT and DLFuzz are used for classification tasks.

Hyperparameter Settings. For FTDG, the default values of hyperparameters are shown in Table 2. We measure the diversity of the generated adversarial test inputs by calculating the sum of the distances [8]. To reduce the impact of randomness on the results, each method is repeated 10 times and the average is calculated as the final experimental result.

Table 2: The hyperparameter values for experiments under each dataset.

Dataset	Hyperparameters					
	s	p	t	λ	m	$iter_{times}$
MNIST	0.02	0.02	0.5	0.5	5	5
SVHN	0.01	0.02	0.5	0.5	5	5
Driving	2.5	0.02	0.5	0.5	5	5

4.3 Results Analysis

Analysis of the Effectiveness of Seed Selection Strategies. In the experiments, FTDF applies the seed selection strategy based on model prediction uncertainty (Uncertainty) and the random seed selection strategy (Random) to construct seed lists and guide the generation of test data. In addition, we measure the diversity of the generated adversarial test inputs by calculating the sum of the L_2 distances between all original seeds and all adversarial test data generated from the corresponding original seeds.

Table 3 shows the average results of each method in improving neuron coverage of the DNN model and generating adversarial test data using different seed selection strategies. The rows "Avg(C)" and "Avg(R)" represent the overall average results of each classification and regression method across all models for the three evaluation metrics. The rows "↑(%)(C)" and "↑(%)(R)" represent the improvement rates when using our method, compared to using the Random. A dash '-' in the table indicates that the method is not applicable. 'Un' represents the seed selection strategy based on model prediction uncertainty, while 'Rd' stands for the random seed selection strategy.

In this experiment, there are a total of 16 classification cases (4 classification models × 4 methods) and 2 regression cases (1 regression model × 2 methods). In the 16 classification cases, the seed selection strategy based on model prediction uncertainty performed the best in improving neuron coverage in 13 cases, and in terms of the number and diversity of adversarial test data in 14 cases. In the

regression cases, the approach Uncertainty performed best on all three metrics. In all cases, including both classification and regression tasks, each method shows an average improvement of 6.7% to 94.7% in neuron coverage, 17.6% to 141.8% in the number of adversarial test data, and 20.3% to 153.4% in the diversity of adversarial test data when using Uncertainty method compared to Random.

Table 3: The results of different methods using different seed selection strategies.

Model / Avg Result / Increase Rate	Seed Strategy	Neuron Coverage (%) / Number of Adv / Diversity of Adv			
		FTDG	DeepXplore	ADAPT	DLFuzz
MNI-1	Un	**46.2/250.0/392.4**	**17.3/20.0/25.5**	**40.3/240.7/286.9**	**40.4/217.0/243.0**
	Rd	40.6/100.0/160.2	15.3/12.0/4.5	36.0/94.7/114.4	36.0/96.2/124.4
MNI-2	Un	**71.2/249.1/385.9**	**34.4/13.0/11.9**	**62.6/203.7/234.2**	**70.2/214.1/233.6**
	Rd	65.7/110.4/145.0	25.8/7.5/7.5	59.2/104.4/120.6	53.9/102.0/133.8
SVH-1	Un	**40.1/255.0/339.7**	32.3/**25.0**/33.6	**39.2/210.7/299.8**	**39.2/211.2/301.1**
	Rd	37.4/95.6/123.9	**32.7/25.0/38.5**	28.2/99.6/107.8	29.8/102.2/124.3
SVH-2	Un	**42.0/248.0/300.0**	36.9/14.0/18.8	45.3/**207.8/230.0**	**40.3/211.3/275.1**
	Rd	38.0/108.5/130.4	**39.2/16.6/24.3**	**48.5**/104.5/135.2	29.9/106.3/146.6
Dave-drop	Un	**49.5/61.2/52194.9**	**25.5/28.0/25867.2**	—	—
	Rd	38.4/41.0/39205.8	13.1/15.0/18329.5	—	—
Avg(C)	Un	**49.9/250.5/354.5**	**30.2/18.0/22.5**	**46.9/215.7/262.7**	**47.5/213.4/263.2**
	Rd	45.4/103.6/139.9	28.3/15.3/18.7	43.0/100.8/118.8	37.4/101.7/132.3
Avg(R)	Un	**49.5/61.2/52194.9**	**25.5/28.0/25867.2**	—	—
	Rd	38.4/41.0/39205.8	13.1/15.0/18329.5	—	—
↑ (%)(C)		9.9/141.8/153.4	6.7/17.6/20.3	9.1/114.0/121.1	27.0/109.9/98.9
↑ (%)(R)		28.9/49.3/33.1	94.7/86.7/41.1	—	—

Therefore, the seed selection strategy based on model prediction uncertainty proposed in this paper is more effective than the random seed selection strategy used in existing research. It is particularly effective in improving model coverage and generating adversarial test data. Thus, it has significant advantages in enhancing the effectiveness of fuzz testing.

Analysis of the Effectiveness of Neuron Selection Strategies. In this experiment, each method is compared under the premise of using Uncertainty to evaluate the impact of their respective neuron selection strategies on the overall effectiveness of the methods. Table 4 shows the results of each method in improving neuron coverage of the DNN model and generating adversarial test data. The rows "↑(%)(C)" and "↑(%)(R)" represent the effectiveness improve-

ment rates of FTDG compared to each classification and regression method in terms of the three evaluation metrics, when using Uncertainty.

Table 4: The effectiveness of different methods in improving neuron coverage.

Model / Avg Result / Increase Rate	Neuron Coverage (%) / Number of Adv / Diversity of Adv			
	FTDG	DeepXplore	ADAPT	DLFuzz
MNI-1	**46.2/250.0/392.4**	17.3/20.0/25.5	40.3/24.7/286.9	40.4/217.0/243.0
MNI-2	**71.2/249.1/385.9**	34.4/13.0/11.9	62.6/203.7/234.2	70.2/214.1/233.6
SVH-1	**40.1/255.0/339.7**	32.3/25.0/33.6	39.2/210.7/299.8	39.2/211.2/301.1
SVH-2	**42.0/248.0/300.0**	36.9/14.0/18.8	45.3/207.8/230.0	40.3/211.3/275.1
Dave-drop	**49.5/61.2/52194.9**	25.5/28.0/25867.2	—	—
Avg(C)	**49.9/250.5/354.5**	30.2/18.0/22.5	46.9/215.7/262.7	47.5/213.4/263.2
Avg(R)	**49.5/61.2/52194.9**	25.5/28.0/25867.2	—	—
↑ (%)(C)	—	65.2/1291.7/1475.6	6.4/16.1/34.9	5.1/17.4/34.7
↑ (%)(R)	—	94.1/118.6/101.8	—	—

Across all classification cases, the neuron coverage of FTDG improves by an average of 5.1% to 65.2% compared to other methods. In comparison with ADAPT and DLFuzz, FTDG increases the number of adversarial test data generated by 16.1% and 17.4%, respectively, and enhances the diversity of adversarial test data by 34.9% and 34.7%, respectively. The effectiveness of DeepXplore is closely related to the choice of models. In the experiments, the set of models used by DeepXplore for classification tasks are functionally very similar, resulting in a small number and low diversity of test data generated to trigger divergent behaviors in the models. Therefore, compared to DeepXplore, FTDG significantly improves the number and diversity of adversarial test inputs generated, with increases of 1291.7% and 1475.6%, respectively. Across the regression cases, compared to other methods, FTDG improves neuron coverage, the number of generated adversarial test data, and diversity by 39.0% to 94.1%, 35.7% to 118.6%, and 63.9% to 101.8%, respectively.

Fig. 2 shows the original images of MNIST, SVHN, and Driving, as well as the corresponding adversarial test data images generated by FTDG. A comparison between the original images and the generated adversarial test data images reveals that the perturbations produced by FTDG are minimal and barely perceptible to the human eye.

Analysis of Hyperparameters Impact on the Effectiveness in Neuron Coverage. In this experiment, to evaluate the impact of different settings of the

Fig. 2: Original images and adversarial test images generated by FTDG.

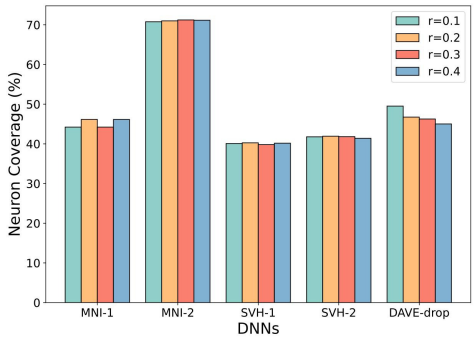


Fig. 3: Neuron coverage results of the FTDG under different values of r.

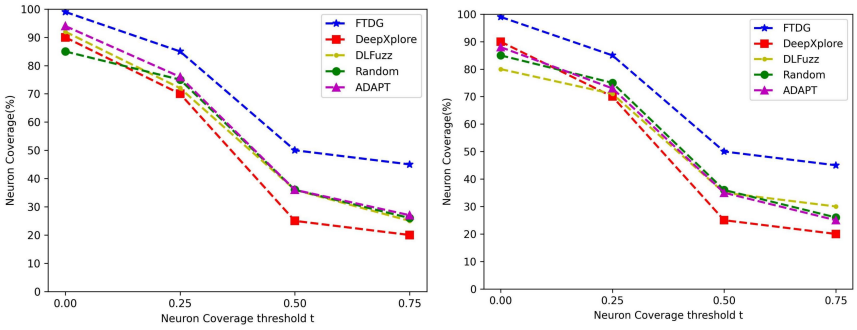
hyperparameter r (the proportion of error neurons) on improving the neuron coverage of FTDG, we set r to 0.1, 0.2, and 0.3, respectively. Fig. 3 shows the results of FTDG in enhancing neuron coverage under different values of r. For different models, different values of r result in different effects on neuron coverage. Thus, for MNI-1, we set the default value of r is 0.2. For MNI-2, we set r=0.3. For SVH-1 and SVH-2, the values of r make less impact on the neuron coverage and FTDG achieves slightly higher neuron coverage at r=0.2, so we set r=0.2 as the default value. For DAVE-drop, we set r=0.1 as the default value.

Additionally, we compare the neuron coverage achieved by each method under different settings of the hyperparameter t (neuron activation threshold) at 0, 0.25, 0.5, and 0.75, and evaluate the impact of t on improving neuron coverage. Fig. 4 shows the results on the MNI-1 and SVH-1 models. From the results, we find that FTDG always achieves higher neuron coverage than other methods under different settings of t. As the threshold t increases, all five approaches cover fewer neurons. This is because a higher threshold t makes it more difficult for minor perturbations to activate neurons.

5 Threats to validity

This section discusses the threats to the validity of the FTDG method, which are mainly reflected in the following aspects.

Internal Validity. The implementations of FTDG and other comparison methods may differ, which could affect the validity of the experimental results. To mitigate this threat, FTDG uses the source code released by the original authors of the comparison methods. Additionally, to ensure the fairness of the experiments, we examine the experimental setup to ensure that, except for the seed selection and neuron selection strategies, all hyperparameters and variable settings remain consistent across the compared methods.



(a) MNI-1 (b) SVH-1

Fig. 4: Neuron coverage results of different methods under different values of t.

External Validity. External validity threats mainly arise from the choice of datasets and DNN models. FTDG uses three publicly available and widely used datasets, along with five popular DNN models with different architectures, ensuring the broad applicability of FTDG across various application scenarios and model structures.

Construct Validity. Random factors may affect the stability of the experimental results. To minimize the impact of randomness, we repeat each experimental configuration 10 times and average the results. Additionally, FTDG evaluates multiple hyperparameter settings and analyzes their impact on the results, reducing the potential threat of parameter settings to the experimental validity.

6 Conclusion

In this paper, we propose FTDG, a test data generation method based on neuron coverage. To effectively enhance the error exposure and test coverage capabilities of fuzz testing, FTDG leverages the output information of test data on DNN models to design a seed selection strategy based on model prediction uncertainty. In the seed mutation strategy, FTDG employs LRP to design a neuron selection strategy based on model prediction error correlations. In conjunction with the neuron selection strategy, FTDG guides the mutation of seeds towards triggering more erroneous behaviors in the DNN model and activating more neurons.

The experimental results show that FTDG performs better in improving neuron coverage, generating the number, and generating the diversity of adversarial test data than existing methods. This indicates that FTDG is more effective in generating test data with high error revelation and improved DNN coverage.

Acknowledgments. This work is supported by the Fundamental Research Funds for the Central Universities(No. NT2024020), the Open Fund of the State Key Laboratory for Novel Software Technology (No.KFKT2024B27), the National Science Foundation of China (No.62402215), and China Postdoctoral Science Foundation (No.2023M741685).

References

1. Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.R., Samek, W.: On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. PloS one **10**(7), e0130140 (2015)
2. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. In: 2017 ieee symposium on security and privacy (sp). pp. 39–57. Ieee (2017)
3. Dai, H., Sun, C.a., Liu, H.: Deepcontroller: Feedback-directed fuzzing for deep learning systems. strategies **5**, 6 (2022)
4. Du, X., Xie, X., Li, Y., Ma, L., Liu, Y., Zhao, J.: Deepstellar: Model-based quantitative analysis of stateful deep learning systems. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 477–487 (2019)
5. Feng, Y., Shi, Q., Gao, X., Wan, J., Fang, C., Chen, Z.: Deepgini: prioritizing massive tests to enhance the robustness of deep neural networks. In: Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis. pp. 177–188 (2020)
6. Goodfellow, I.J.: Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014)
7. Gopinath, D., Zhang, M., Wang, K., Kadron, I.B., Pasareanu, C., Khurshid, S.: Symbolic execution for importance analysis and adversarial generation in neural networks. In: 2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE). pp. 313–322. IEEE (2019)
8. Guo, J., Jiang, Y., Zhao, Y., Chen, Q., Sun, J.: Dlfuzz: Differential fuzzing testing of deep learning systems. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 739–743 (2018)
9. Han, S., Lin, C., Shen, C., Wang, Q., Guan, X.: Interpreting adversarial examples in deep learning: A review. ACM Computing Surveys **55**(14s), 1–38 (2023)
10. Hu, X., Jin, J., Xiong, M., Liu, J., Peng, T., Zhang, Z., Chen, J., He, R., Qin, X.: Multi-scale gated inpainting network with patch-wise spacial attention. In: Database Systems for Advanced Applications. DASFAA 2021 International Workshops: BDQM, GDMA, MLDLDSA, MobiSocial, and MUST, Taipei, Taiwan, April 11–14, 2021, Proceedings 26. pp. 169–184. Springer (2021)
11. Jia, Y., Harman, M.: An analysis and survey of the development of mutation testing. IEEE transactions on software engineering **37**(5), 649–678 (2010)
12. Kendall, A., Gal, Y.: What uncertainties do we need in bayesian deep learning for computer vision? Advances in neural information processing systems **30** (2017)
13. Kurakin, A., Goodfellow, I.J., Bengio, S.: Adversarial examples in the physical world. In: Artificial intelligence safety and security, pp. 99–112. Chapman and Hall/CRC (2018)
14. LeCun, Y.: The mnist database of handwritten digits. http://yann. lecun. com/exdb/mnist/ (1998)
15. Lee, S., Cha, S., Lee, D., Oh, H.: Effective white-box testing of deep neural networks with adaptive neuron-selection strategy. In: Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis. pp. 165–176 (2020)
16. Li, Y.: Research and application of deep learning in image recognition. In: 2022 IEEE 2nd international conference on power, electronics and computer applications (ICPECA). pp. 994–999. IEEE (2022)

17. Liu, Z., Feng, Y., Chen, Z.: Dialtest: automated testing for recurrent-neural-network-driven dialogue systems. In: Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis. pp. 115–126 (2021)
18. Majumdar, R., Sen, K.: Hybrid concolic testing. In: 29th International Conference on Software Engineering (ICSE'07). pp. 416–426. IEEE (2007)
19. Manès, V.J., Han, H., Han, C., Cha, S.K., Egele, M., Schwartz, E.J., Woo, M.: The art, science, and engineering of fuzzing: A survey. IEEE Transactions on Software Engineering **47**(11), 2312–2331 (2019)
20. Moosavi-Dezfooli, S.M., Fawzi, A., Frossard, P.: Deepfool: a simple and accurate method to fool deep neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2574–2582 (2016)
21. Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A.Y., et al.: Reading digits in natural images with unsupervised feature learning. In: NIPS workshop on deep learning and unsupervised feature learning. vol. 2011, p. 4. Granada (2011)
22. Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A.: The limitations of deep learning in adversarial settings. In: 2016 IEEE European symposium on security and privacy (EuroS&P). pp. 372–387. IEEE (2016)
23. Papineni, K., Roukos, S., Ward, T., Zhu, W.J.: Bleu: a method for automatic evaluation of machine translation. In: Proceedings of the 40th annual meeting of the Association for Computational Linguistics. pp. 311–318 (2002)
24. Pei, K., Cao, Y., Yang, J., Jana, S.: Deepxplore: Automated whitebox testing of deep learning systems. In: proceedings of the 26th Symposium on Operating Systems Principles. pp. 1–18 (2017)
25. Sun, Y., Wu, M., Ruan, W., Huang, X., Kwiatkowska, M., Kroening, D.: Concolic testing for deep neural networks. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. pp. 109–119 (2018)
26. Szegedy, C.: Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199 (2013)
27. Tian, Y., Pei, K., Jana, S., Ray, B.: Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In: Proceedings of the 40th international conference on software engineering. pp. 303–314 (2018)
28. Udacity: Self-driving car repository. `https://github.com/udacity/self-driving-car`, last accessed: 18 Nov. 2024
29. Xia, Y., Liu, S., Hu, R., Yu, Q., Feng, X., Zheng, K., Su, H.: Smart: A decision-making framework with multi-modality fusion for autonomous driving based on reinforcement learning. In: International Conference on Database Systems for Advanced Applications. pp. 447–462. Springer (2023)
30. Xie, X., Ma, L., Juefei-Xu, F., Xue, M., Chen, H., Liu, Y., Zhao, J., Li, B., Yin, J., See, S.: 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 (2019)
31. Zhang, P., Dai, Q., Ji, S.: Condition-guided adversarial generative testing for deep learning systems. In: 2019 IEEE International Conference on Artificial Intelligence Testing (AITest). pp. 71–72. IEEE (2019)
32. Zhang, P., Ren, B., Dong, H., Dai, Q.: Cagfuzz: coverage-guided adversarial generative fuzzing testing for image-based deep learning systems. IEEE Transactions on Software Engineering **48**(11), 4630–4646 (2021)
33. Zhu, J.Y., Park, T., Isola, P., Efros, A.A.: Unpaired image-to-image translation using cycle-consistent adversarial networks. In: Proceedings of the IEEE international conference on computer vision. pp. 2223–2232 (2017)