DistXplore: Distribution-Guided Testing for Evaluating and Enhancing Deep Learning Systems

Longtian Wang
Xi'an Jiaotong University
China

Xiaofei Xie[*]
Singapore Management University
Singapore

Xiaoning Du
Monash University
Australia

Meng Tian
Singapore Management University
Singapore

Qing Guo
IHPC and CFAR, Agency for Science, Technology and Research
Singapore

Zheng Yang
TTE Lab, Huawei
China

Chao Shen[*]
Xi'an Jiaotong University
China

ABSTRACT

Deep learning (DL) models are trained on sampled data, where the distribution of training data differs from that of real-world data (*i.e.*, the distribution shift), which reduces the model's robustness. Various testing techniques have been proposed, including distribution-unaware and distribution-aware methods. However, distribution-unaware testing lacks effectiveness by not explicitly considering the distribution of test cases and may generate redundant errors (within same distribution). Distribution-aware testing techniques primarily focus on generating test cases that follow the training distribution, missing out-of-distribution data that may also be valid and should be considered in the testing process.

In this paper, we propose a novel distribution-guided approach for generating *valid* test cases with *diverse* distributions, which can better evaluate the model's robustness (*i.e.*, generating hard-to-detect errors) and enhance the model's robustness (*i.e.*, enriching training data). Unlike existing testing techniques that optimize individual test cases, *DistXplore* optimizes test suites that represent specific distributions. To evaluate and enhance the model's robustness, we design two metrics: *distribution difference*, which maximizes the similarity in distribution between two different classes of data to generate hard-to-detect errors, and *distribution diversity*, which increase the distribution diversity of generated test cases for enhancing the model's robustness. To evaluate the effectiveness of *DistXplore* in model evaluation and enhancement, we compare *DistXplore* with 14 state-of-the-art baselines on 10 models across 4 datasets. The evaluation results show that *DistXplore* not only detects a larger number of errors (*e.g.*, 2×+ on average), but also identifies more hard-to-detect errors (*e.g.*, 10.5%+ on average); Furthermore, *DistXplore* achieves a higher improvement in empirical robustness (*e.g.*, 5.2% more accuracy improvement than the baselines on average).

[*]Corresponding authors

ACM ISBN 979-8-4007-0327-0/23/12.
https://doi.org/10.1145/3611643.3616266

CCS CONCEPTS

• Software and its engineering → Software testing and debugging; **• Computing methodologies → Neural networks**.

KEYWORDS

Deep learning, software testing, distribution diversity, model enhancement

ACM Reference Format:
Longtian Wang, Xiaofei Xie, Xiaoning Du, Meng Tian, Qing Guo, Zheng Yang, and Chao Shen. 2023. *DistXplore*: Distribution-Guided Testing for Evaluating and Enhancing Deep Learning Systems. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '23), December 3–9, 2023, San Francisco, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3611643.3616266

1 INTRODUCTION

Deep learning (DL) has achieved great success in many applications such as autonomous driving [42], healthcare [47], face recognition [18] and speech recognition [68]. It is widely known that DL models suffer from the issue of poor robustness, making them vulnerable to adversarial attacks. Therefore, it is crucial to systematically test DL systems before deployment, especially in safety-critical scenarios.

Machine learning (ML) involves the process of learning a model from sampled data (*i.e.*, training data) to make decisions on a specific task. The general steps of ML tasks include data collection, model training, model evaluation, and model deployment. Due to the huge input space, it is impossible to collect all data for training, thus, high-quality data that follows a certain distribution is collected for training. As shown in Fig. 1, for a specific task (*e.g.*, digit classification), there is a vast amount of task-relevant data for digits (*i.e.*, the *valid* data shown in the dashed rectangle) in the whole input space (*i.e.*, *all* data shown in the solid rectangle). The task-irrelevant data (*e.g.*, noisy data and non-digit data) is referred to as *invalid* data (*e.g.*, the dataset f in Fig. 1) with respect to the

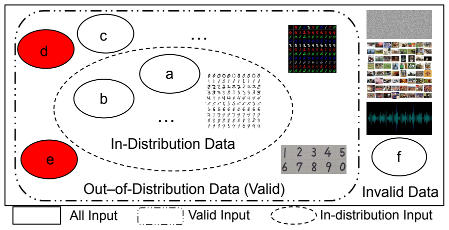

Figure 1: Data sampling and an illustrative example of DL system

given task. A small subset of the valid data (*e.g.*, the dataset *a* and *b* in Fig. 1) is collected for training the model. However, the training distribution is often different from the distribution of valid data (due to the distribution shift), which greatly affects the model's robustness. A fundamental assumption is that the model is intended to handle the in-distribution data (ID) that follows the distribution of training data [4], but it is hard to correctly predict data (*e.g.*, the dataset *c*, *d*, and *e* in Fig. 1) that does not follow the training distribution, *i.e.*, out-of-distribution data (OOD), which highlights the need for testing before deployment.

DL testing aims to generate test cases that *evaluate* the robustness of DL systems, *i.e.*, discover the data that is valid but cannot be predicted correctly (*e.g.*, the dataset *d* and *e* in Fig. 1), and *enhance* the robustness, *i.e.*, retraining model by including test cases data with diverse distribution (dataset *c*, *d*, and *e* in Fig. 1). Many studies have been conducted for testing DL systems [9, 22, 46, 54, 57, 63], where *validity* and *distribution* are two important properties of test cases. A common approach to guarantee *validity* is to constrain the degree of the mutation (*e.g.*, the distance between the new test and the original seed is constrained within a L_p ball). However, existing methods (*e.g.*, DeepTest [54], DeepHunter [63], and TensorFuzz [43]) often ignore the distribution [4, 9], which limits their effectiveness in evaluation and enhancement (*e.g.*, redundant errors within the similar distribution are generated). Recently, some studies [4, 9, 22, 55] have attempted to address this by incorporating distribution-aware testing, which characterizes the training distribution via Variational Auto-Encoder (VAE) or Generative Adversarial Network (GAN). However, these methods only generate ID data while OOD data is considered as "invalid". We argue that the OOD data is just data that does not follow the distribution of the collected training data but could still be valid and should be handled properly in real-world deployment environment. For example, as shown in Fig. 2, for each dataset, the input on the right side in a row is mutated from its left-side sample, the inputs on the right are considered as "invalid" data by existing distribution-aware testing [9]. These data could still be visually valid, even though they are identified as "invalid" data by VAE [9]. For example, although the distribution of the images in *Out-of-Distribution Data (Valid)* in Fig. 1 is different from the distribution of the training data (*e.g.*, the digits written in very different ways), they could still be the potential inputs to the deployed DL systems. Therefore, it is crucial to test both in-distribution (ID) and out-of-distribution (OOD) data that are valid before deploying the DL system.

The quality of test cases depends on the testing goals, *i.e.*, what kind of data is more useful in robustness evaluation and enhancement in this paper. For evaluating model's robustness, although OOD data is likely to trigger incorrect decisions of the model, they

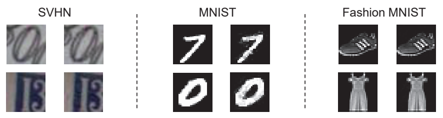

Figure 2: Examples of OOD data that are considered as *invalid* by [9]. Left: original inputs, Right: generated inputs

could also be easily detected by OOD detection methods. For example, state-of-the-art testing techniques can easily generate a large number of errors (*e.g.*, thousands of errors in [46, 63]), but most of them tend to be weak errors that can be detected or filtered by existing defense techniques (*e.g.*, adversarial example detection [58]). It is similar to traditional software testing, where defenses such as parsers and exception handling can filter out weak errors. Thus, for DL testing, it is important and challenging to discover strong errors that can evade the state-of-the-art defenses. For model enhancement, the general goal is to reduce the distribution shift between the training data and real-world data. Hence, how to generate tests with diverse distributions (*e.g.*, covering *c*, *d*, *e*) is another challenge. These diverse tests can be added to the training data for improving the model generalizability and robustness.

To this end, in this paper, we propose a novel distribution-guided testing framework (named *DistXplore*) for better *evaluating* and *enhancing* DL systems, *i.e.*, to generate *hard-to-detect* and *diverse* errors. *DistXplore* adopts the search-based approach to adaptively generate test cases with the guidance of distribution. Unlike existing techniques that optimize test cases individuality, the optimization of *DistXplore* is performed on a test suite that represents a specific distribution. Specifically, we leverage Maximum Mean Discrepancy (MMD) [13] to measure the closeness between two distributions. For model evaluation, *DistXplore* maximizes the distribution closeness between the data in two different classes for generating statistically indistinguishable errors, which are difficult to defend. To enhance the model's robustness, we propose a metric to measure the distribution diversity of the test cases, guiding *DistXplore* to generate test suites with various distributions. The test cases with diverse distributions are more likely to cover a wider range of unseen data and improve the model's robustness.

We conduct a comprehensive evaluation to demonstrate the usefulness and the effectiveness of *DistXplore* in evaluating and enhancing the model's robustness. Specifically, we select 10 models on 4 datasets, and compare *DistXplore* with 14 state-of-the-art tools covering 4 different types of techniques (*i.e.*, adversarial attacks, distribution-unaware testing, distribution-aware testing, and robustness-oriented testing). The results demonstrate that 1) the statistically indistinguishable errors generated by *DistXplore* are harder to detect by two state-of-the-art defense techniques, *e.g.*, the attack-as-defense [69] can only detect 66% errors generated by *DistXplore*, but almost 100% errors from adversarial attacks and distribution-aware testing. 2) *DistXplore* is more efficient in detecting errors, *e.g.*, on average it detects 2×+ errors compared to the best baseline. 3) The test cases generated by *DistXplore* are more useful in improving the model's robustness, *e.g.*, 5.2% more accuracy improvement than the baselines on average.

To summarize, this paper makes the following contributions:

- We first discuss the limitation of existing distribution-aware and distribution-unaware testing techniques in terms of validity and

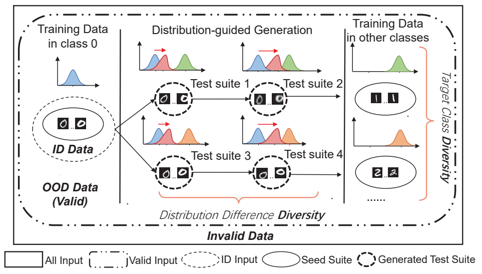

Figure 3: Illustration of test suite generation

distribution. Then we propose a novel distribution-guided testing technique for generating hard-to-detect errors and diverse data covering a wider range of unseen data. To the best of our knowledge, this is the first distribution-guided testing for generating test suites with diverse distributions.

- Technically, we design two distribution-based metrics (*i.e.*, distribution difference and distribution diversity) to guide the testing for generating statistically indistinguishable errors and test cases with diverse distributions, respectively.
- We demonstrate the usefulness of *DistXplore* in discovering strong errors and enhancing model's robustness by comparing it with 14 state-of-the-art methods.

2 PRELIMINARY AND OVERVIEW

2.1 Preliminary

2.1.1 Deep Neural Network. A Deep Neural Network (DNN) can be represented as a function $f : X \longrightarrow Y$ that maps an n-dimensional input $x \in X$ to an m-dimensional output $y \in Y$. A DNN usually is the composition of layers denoted as $f = l_0 \circ l_1 \circ \ldots \circ l_k$. We use $f_i(x)$ to represent the output of the i^{th} layer, where $f_0(x) = x$ and $f_k(x) = y$. For example, the output y in classification is a probability vector for m possible classes (*e.g.*, 10 classes in CIFAR-10).

2.1.2 Data Validity. Let X be the whole input space with n dimensions (*i.e.*, $\mathbb{R}^n$). We use Z to denote all possible inputs that are relevant to the given task (*e.g.*, all images of digits 0-9). Z is considered as valid data with respect to the task as they could be the potential inputs when the trained model is deployed in real-world. The inputs $X \backslash Z : \{x : x \in X \wedge x \notin Z\}$ are invalid data, *e.g.*, the data of other tasks and low-quality data. It is difficult to precisely define the validity of the data. In practice, the L_p norm [40] is usually used to guarantee the validity of the generated data by the existing DL testing and adversarial attack techniques. Specifically, given a valid input x, the new test case x' generated by adding some perturbations on x is considered as valid if $||x' - x||_p < d$, where d is a safe radius.

2.1.3 Data Distribution. Since valid inputs Z can be infinite, it is not possible to collect all of them for training. In practice, a DNN f is usually trained from collected data T (*i.e.*, training data) that follows a distribution $\mathcal{D}_T$, called in-distribution (ID) data. Some generative models such as variational autoencoders (VAE) [30] and generative adversarial networks (GAN) [9] are used to approximate the ID data distribution [22].

There is often a distribution shift between $\mathcal{D}_Z$ and $\mathcal{D}_T$ (*i.e.*, the training data cannot represent the real-world data), making that the model underperforms on the out-of-distribution (OOD) data. Hence, test cases with diverse distributions are more likely to reveal the weaknesses of the model. On the other hand, the OOD test cases can enrich the training data such that the distribution of new training dataset is closer to the distribution of training data.

Note that the validity and the out-of-distribution of the data are different in this paper. The valid data is *any* potential inputs of the model with respect to the task, and is usually of high quality. The out-of-distribution data refers to the data that does not follow the distribution of specific training data. The valid data can be ID or OOD, depending on the training data collected. The OOD data can also be valid or invalid, depending on the relevance and quality of the data. To measure the validity, we adopt the widely used measurement, *i.e.*, L_p norm. To measure the distribution difference, we adopt the metric Maximum Mean Discrepancy defined below.

2.1.4 Maximum Mean Discrepancy. Maximum Mean Discrepancy (MMD) is a common test statistic to measure the closeness between two sets of samples drawn from two distributions. Assume we have two sets of samples $X = \{x_1, \ldots, x_m\}$ and $Y = \{y_1, \ldots, y_n\}$ drawn from two distributions $\mathcal{D}_X$ and $\mathcal{D}_Y$, MMD calculates the distance between the two sets of samples in a universal reproducing kernel Hilbert space (RKHS) [51]. The empirical estimation of MMD between the two distributions in RKHS, denoted as $MMD(X, Y)$, can be calculated as:

$$\frac{1}{m^2}\sum_{i,j=1}^{m} k(x_i, x_j) - \frac{2}{mn}\sum_{i,j=1}^{m,n} k(x_i, y_j) + \frac{1}{n^2}\sum_{i,j=1}^{n} k(y_i, y_j)$$

where k is a measurable and bounded kernel of a RKHS, MMD is zero if and only if $\mathcal{D}_X = \mathcal{D}_Y$. As mentioned in [44] that Gaussian and Laplace kernels are universal, we use Gasussian kernel to calculate MMD. More details about MMD can refer to [14].

2.2 Overview of *DistXplore*

Fig. 3 shows the main idea of our approach. We mainly consider classification task in this paper. Specifically, *DistXplore* considers the data distribution in each class separately, *i.e.*, to generate test cases with diverse distributions for each class. To measure the distribution diversity of the test cases, we calculate the distribution difference (*i.e.*, MMD) between the test suite from a class and the training data in each of other classes, and then measure the diversity of these distribution differences. We consider distribution differences between test cases and the data in different classes, since each input may be classified into any class by a model, representing the different decision behaviors of the model. Therefore, we aim to generate diverse test cases by considering the diversity of distribution differences between the generated test cases and training data of different classes.

As shown in Fig. 3, given the initial test suite sampled from the training data of a class, which represents the training distribution of the class, the goal is to generate new test suites that have different distribution distances with the training data in other classes (*e.g.*, class 1, 2, 3). The distribution curve of the test suites (*i.e.*, red curve) shifts from the original distribution (*i.e.*, blue curve) to the target distribution (*i.e.*, green or orange curve), thus *DistXplore* generates test suites that are more likely to be predicted incorrectly. For robustness evaluation, the goal is to generate errors that are hard to

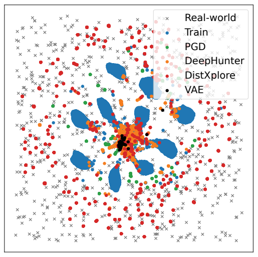

Figure 4: Diversity of data distribution on MNIST

detect. The more similar the distribution of the test suite (*e.g.*, class 0) is to the distribution of the training data in the target class (*e.g.*, class 1), the harder it is to detect the errors, because the errors are statistically indistinguishable from the target class. For robustness enhancement, *DistXplore* is used to generate test suites with diverse distributions (instead of only hard-to-defend errors) that can enrich the training data by adding unseen data, thus improving the model's robustness.

3 DISTRIBUTION-GUIDED TESTING

3.1 Testing Goals

In this paper, we mainly focus on two objectives: model evaluation and enhancement. We design the objective functions that guide the test case generation, *i.e.*, optimize test suites.

3.1.1 Model Evaluation. To evaluate the model's robustness, we aim to generate the erroneous inputs that are hard to be detected by existing defense techniques. Specifically, just as any dataset can follow a specific distribution, the data within individual classes in classification tasks also possess their own distribution. Due to the differences between these different classes, their data distributions are also very different (*e.g.*, dogs and birds). A well-trained model is capable of accurately distinguishing the differences between these classes. In Fig. 4, the blue areas represent the different distributions of data for different classes in the MNIST dataset. Conversely, if the data distributions between two classes are very similar, the model may struggle to make accurate predictions. Thus, *DistXplore* aims to generate test cases (in a class) that are statistically similar to the training data in other classes.

Formally, given a DNN f and a test suite S_c belonging to a source class c, we define its distribution difference with respect to the training data ($T_{c'}$) in another target class c' as:

$$DF_f(S_c, c') = MMD(f_l(S_c), f_l(T_{c'}))$$

where f_l refers to the output of the layer l and $c' \neq c$.

The distribution difference is measured on a specific layer of the DNN. In this paper, we select the logits layer, *i.e.*, the layer before the softmax layer, which is frequently used in previous works [26, 31, 70]. Intuitively, the smaller the value $DF_f(S_c, c')$, the more difficult it is for the model f to distinguish S_c and $T_{c'}$. Hence, it is more likely to generate undetectable errors by minimizing their distribution difference.

3.1.2 Model Enhancement. The model's robustness can be improved if the distribution of training data (T) is closer to the distribution of real-world valid data (Z), *i.e.*, to add more unseen valid data to training data. However, it is impossible to directly collect all real-world data. Therefore, we could adjust the objective to generate data that is as diverse as possible, aiming to make the distribution of the generated data more closely resemble that of real-world valid data (Z). To provide a easy understanding of the fundamental concept behind generating diverse data to enhance model's robustness, we conducted a qualitative analysis, as depicted in Fig. 4. In this visualization, we show the distribution of training data (represented in blue) and the distributions of specific errors generated by different types of tools: adversarial attack tool (PGD [37]), distribution-unaware testing tool (DeepHunter [63]), distribution-aware testing tool (VAE [55]), and *DistXplore*. Additionally, we include some real-world data examples, which represent a wide range of possible data samples.

The results of this analysis highlight two key observations: 1) the model is not robust due to the distribution shift between the training data and real-world data. By utilizing various tools, we can generate valid OOD data that helps reduce the distribution shift, and further enhance the robustness by incorporating previously unknown data into the training set. 2) The erroneous inputs generated by existing tools exhibit limited diversity, while *DistXplore* aims to generate test cases with diverse distributions, such that the distribution of the generated data could be closer to real-world data distribution.

We propose a metric to measure the distribution diversity of test suites, which can guide the generaton of diverse data. Given a DNN f that performs the classification on m classes (denoted as C_f), and a set of test suites TS_c in a class c, the distribution diversity is defined as:

$$Div(TS_c) = \frac{\sum_{c' \in C_f \backslash c} |\{\mathcal{B}(DF_f(S, c')) | \forall S \in TS_c)\}|}{|C_f \backslash c| \cdot k}$$

where $C_f \backslash c$ represents the other classes except c, $\mathcal{B}$ is an interval abstraction function that maps a concrete MMD value to an interval, and k is the number of intervals between c and each of other classes.

The basic idea is to measure the diversity of distribution differences between the current test suites and the training data of other classes. Since the difference between two distributions (*i.e.*, MMD) is a continuous variable, we adopt the interval abstraction to spilt its values into k intervals (*i.e.*, k distributions). The numerator and the denominator represent the number of intervals covered and the total number of intervals between the current class c and other classes, respectively. As shown in Fig. 3, the distribution adequacy is measured from two perspectives: 1) *Distribution Difference Diversity*: for a given target class c', multiple intervals between the test suites and the training data of c' can be covered. 2) *Target Class Diversity*: multiple classes (*i.e.*, $C_f \backslash c$) are used to guide the test generation, which allows to consider the relationships between every two classes.

Intuitively, the test suites in multiple intervals have different distributions. To enhance the model's robustness, the training data should cover the distributions as many as possible, *i.e.*, to increase the distribution diversity. Note that, only using the strong errors (*i.e.*, undetectable) is not sufficient to improve the whole robustness as it cannot handle errors with different distributions (see the results in Section 4). In Fig. 3, the generated test suites (*i.e.*, *Test suite 1, 2, 3, 4*) have diverse distributions (*i.e.*, different red curves), and are added into the training data for retraining.

We select the classes in the same task as targets because the classification is based on their relationships, *i.e.*, to choose a relatively suitable class (with higher probability). These targets may be incomplete in terms of characterizing the distribution diversity. We can also select other targets to guide the test generation such as the classes in other tasks as long as the generated tests are valid. For example, we can select the classes in CIFAR-10 or Roman numerals as the targets of MNIST task. We plan to evaluate the effects of more different targets in the future work.

3.2 Distribution-Guided Test Generation

To achieve both testing goals, we use a genetic algorithm (GA) to solve the problem. Without loss of generality, the objective function can be defined as $DF_f(S_c, c') \approx v$ (*i.e.*, to decrease $DF_f(S_c, c')$ until it is close to a small value v), where S_c is the test suite belonging to c, c' is a target class and v is a constant value. For the goal of model evaluation, v is set as 0, *i.e.*, to generate S_c that is statistically indistinguishable from the training data in c'. For the goal of model enhancement, *DistXplore* generates test suites that cover more diverse intervals. Consider a target interval $[v_0, v_1]$ that we aim to cover, the objective function is defined as $DF_f(S_c, c') \approx v$, where $v \in [v_0, v_1]$ can be any value within the range. The general objective function can be:

$$\arg\min_{S_c} |DF_f(S_c, c') - v|$$

Algorithm 1 shows the search-based method to solve the objective function. The inputs include the DNN f, a seed test suite S_c from class c, a target class c' ($c' \neq c$) and the target distribution difference v. The output is the new test suite that can reach the target distribution difference. The seed test suite can be collected from training dataset or testing dataset. We first construct a population that contains m test suites (Line 1-3) by mutating the seed test suite m times. Note that the chromosome is a test suite (including multiple inputs) instead of a single input. It repeatedly optimizes the population (Line 4-16) for minimizing the distribution difference. In each iteration, we first calculate the fitnesses of the updated population (Line 6). Then we update the new population with the standard crossover and mutation. If the best chromosome S in the population satisfies the objective or timeout, then the optimization process terminates (Line 9-10). The distribution difference decreases during optimization until it is less than a pre-defined value ϵ. For example, $\epsilon = 0$ indicates that $DF_f(S, c')$ is equal to v. Note that $DF_f(S, c')$ is decreasing for the two test goals, because the distribution of the initial test suite is often far from the distribution of the training data in the target class c'.

We keep the chromosome that has the best fitness unchanged (*i.e.*, no crossover or mutation) to ensure that the optimization does not get worse (Line 11). For others, we first select two chromosomes based on the tournament strategy [39] (Line 13- 14). A uniform crossover is performed between the selected two chromosomes in the input level, *i.e.*, genes in a chromosome are inputs of the model f (Line 15). Each gene in the chromosome S can be selected to mutate with a selection probability r (Line 16).

In this paper, we mainly focus on image classification tasks. *DistXplore* can be easily extended to other domains. We select the diverse image transformations (*e.g.*, translation, rotation, brightness)

Algorithm 1: Test generation

```
Input  : $f$: the target DNN, $S_c$: a seed test suite from class $c$, $c'$:
         the target class, $v$: target distribution difference
Output : $S'_c$: the new test suite
Const  : $m$: population size, $t$: tournament size, $r$: mutation rate
1  $Pop := \emptyset$;
2  for $i \in [0, m)$ do
3      $Pop := Pop \bigcup mutate\_each(S_c)$;
4  while True do
5      for $S \in Pop$ do
6          $fit_S = DF_f(S, c') - v$;
7      for $S \in Pop$ do
8          if $\forall O \in Pop.fit_S \leq fit_O$ then
9              if $fit_S \leq \epsilon$ or timeout then
10                 return S;
11             continue;
12         else
13             $S_1 := tour\_select(Pop, t)$;
14             $S_2 := tour\_select(Pop, t)$;
15             $S := crossover(S_1, S_2)$;
16             $S := mutate\_prob(S, r)$;
```

used in DeepTest [54] and DeepHunter [63]. For each selected gene, the *mutation* randomly selects a transformation function to mutate it. To guarantee the validity of the generated inputs, we adopt the conservative strategy [63] that constrains the transformation with both L_0 and L_∞.

4 EVALUATION

We have implemented *DistXplore* in Python 3.6 based on DL framework Keras (ver.2.3.1) with Tensorflow (ver.1.15.2). To evaluate the effectiveness of *DistXplore* in the model evaluation and model enhancement, we aim to answer the following research questions (RQs), where RQ1 and RQ2 are to demonstrate the effectiveness in model evaluation, RQ3 and RQ4 are to evaluate the model enhancement, and RQ5 is to study the generalization of *DistXplore*.

- **RQ1**: How effective is *DistXplore* in detecting errors [1] that can bypass the defense methods?
- **RQ2**: How efficient is *DistXplore* for discovering valid errors?
- **RQ3**: How effective is *DistXplore* in improving the robustness of the DL model under testing?
- **RQ4**: How useful are distribution difference diversity and target class diversity in improving robustness?
- **RQ5**: Can *DistXplore* be generalized to other domains?

4.1 Setup

4.1.1 Datasets and DNN Models. We select four datasets (*i.e.*, MNIST, Fashion-MNIST, CIFAR-10, and SVHN) and six DNNs (*i.e.*, LeNet-4, LeNet-5, VGG16, ResNet-20, Inception-v3, and Inception-ResNet-v2) that are commonly used in existing works [11, 16, 20, 33, 55, 60, 61].

4.1.2 Baselines. To evaluate the effectiveness of *DistXplore*, we select 4 *types* of approaches including 14 state-of-the-art baselines for the comparisons: 6 adversarial attacks, 4 distribution-unaware

[1]The error in the paper refers to the erroneous inputs that are missclassified.

testing techniques, 3 distribution-aware testing techniques and 1 robustness-oriented testing.

- *Adversarial Attack.* We select 6 adversarial attack techniques, including 3 classical ones *i.e.*, BIM [29], PGD [37], and C&W [5], and 3 new ones, *i.e.*, DI-2-FGSM (D2F) [60], SI-NI-FGSM (SNF) [33], and TI-FGSM (TIF) [11] to generate adversarial examples and compare them with the errors generated by *DistXplore*.
- *Distribution-unaware Testing.* We select DeepHunter [63], Neuron Path Coverage (NPC) [62], and Combinatorial Testing (CT) [50] as the baselines. DeepHunter is configured with two different coverage guidance, *i.e.*, k-multisection Neuron Coverage (KMNC), Neuron Boundary Coverage (NBC). KMNC and NBC are designed to test the major function region and the corner-case region [35]; NPC is configured with Structure-based Neuron Path Coverage (SNPC), which is designed to test the decision logic; CT takes the relationships between neurons in adjacent layers into consideration when testing DNN models.
- *Distribution-aware Testing.* We select three recent distribution-aware testing techniques [22, 27, 55] as baselines. In [27], the test selection criteria are proposed to measure the Surprise Adequacy (SA) of test cases. We select the Likelihood-based SA (LSA) that measures the training distribution with Kernel Density Estimation as a baseline. In [55], a variational auto-encoder (VAE) is used to specifically generate in-distribution test cases. In [22], a hierarchical distribution-aware (HDA) testing is proposed based on the global distribution and local distribution. We denote these two baselines as VAE and HDA, respectively.
- *Robustness-oriented Testing.* To evaluate the robustness enhancement, we select the state-of-the-art robustness-oriented testing technique Robot [57] as our baseline.

4.1.3 Defense Methods. To evaluate the strengths of generated errors by different techniques, we select two state-of-the-art defense methods that detect adversarial examples as follows:

- *Dissector* [56], which dissects the outputs of intermediate layers and calculates a score for the given input. The score shows the degree of similarity between the input and benign data. For LeNet-4 and LeNet-5, we select the fully connected layers. For efficiency, five intermediate layers are selected for larger model. The details are provided on our website [2].
- *Attack as Defense (A2D)* [69], which detects adversarial samples based on the observation that adversarial samples are less robust than benign ones. It measures the robustness of the given inputs with existing adversarial attacks. We use JSMA [45] (that is different from baseline adversarial attacks) to calculate the attack cost of each input for detecting whether it is abnormal input.

4.1.4 Experiment Setup.

Seed Selection. For each task, we randomly select 100 seed inputs for each class from training dataset. Totally, we select 1,000 seeds that are used by all baselines. Note that the HDA approach proposes a distribution-aware strategy to select seeds, hence we configure HDA with two initial seed construction strategies: 1) using the same 1,000 seed inputs as used for other baselines for a fair comparison (denoted as HDA) and 2) using the HDA's own seed selection to select 1,000 initial seed inputs (denoted as HDA_o).

Configuration of DistXplore. We use the 100 initial seeds selected in each class as a seed test suite. For each class c, we run *DistXplore* multiple times (*i.e.*, 9) by setting different target classes c' with Algo. 1. Finally, for each model, we run *DistXplore* 90 times (*i.e.*, 10 source classes × 9 target classes). We set the fitness function as minimizing the distribution difference (*i.e.*, the values of v and ϵ in Algo. 1 are configured as 0). Note that, to calculate the difference efficiently, we randomly select another 100 samples from the training data in class c' instead of all of them. We found that the distribution distance between the selected samples and the corresponding class of training data is close to zero (MMD), which indicates that the selected training samples can represent the distribution of the whole training data.

For each run of *DistXplore*, we limit the total number of iterations in GA as 30. We empirically configured the population size, the tournament size, and the mutation rate as as 100, 20, and 0.01, respectively. Due to the limit of the space, the experiments about the impact of the parameters are put on our Website [2]. For the robustness enhancement, we do not explicitly generate test cases for each interval (see $Div(TS_c)$ in Section 3.1.2). Instead, we map the distribution difference in each iteration (*i.e.*, the fitness value) to an interval. During the optimization process, the distribution distance is decreasing in multiple iterations, covering different intervals. To ensure the validity of the generated test cases, we adopt a more conservative configuration compared to DeepHunter [63] to constrain the mutation.

Configuration of Baselines. For the three classic adversarial attacks, we perform the target attack for each seed input by selecting other classes as the targets, *i.e.*, we generate 9 adversarial examples for each seed input. For the three new adversarial attacks, as they are not designed for target attacks, we perform untarget attack with the default configurations provided.

Note that LSA is a test selection metric instead of a testing tool. To perform the comparison, we develop a new testing tool based on DeepHunter, *i.e.*, using LSA as the guidance to generate test cases.

For others, we follow their default configurations to run DeepHunter, CT, NPC, HDA, VAE, and Robot. Specifically, each model is tested for 5,000 iterations by DeepHunter (KMNC and NBC), CT, and NPC. Each seed is optimized with 50, 30, and 30 iterations by HDA, VAE, and Robot, respectively. More detailed settings can be found on our website [2].

RQ Setup. To demonstrate the capability of *DistXplore* in generating strong errors for model evaluation (**RQ1**), we collect the test suite in the last iteration for every pair (c, c') (*i.e.*, the best chromosome returns from Algo 1). For each model, we collect a total number of 90 chromosomes over 90 pairs, which are used to evaluate the strength of these errors. The strength of errors is measured by the success rate of bypassing defenses. In addition, we also evaluate the efficiency of *DistXplore* for discovering valid errors (**RQ2**). To evaluate the efficiency, we count all the errors generated during the 30 iterations. Specifically, we select two metrics for the comparisons: the total number of errors and the success rate of generating errors for each seed. To evaluate the validity of generated errors, we perform a human study to manually check the validity of the discovered errors.

Table 1: Results of bypassing the defense techniques on datasets MNIST (M), Fashion MNIST (FM), CIFAR-10 (C), and SVHN (S) and DNNs LeNet-4 (L-4), LeNet-5 (L-5), VGG16 (V-16), ResNet-20 (R-20), Inception-ResNet-v2 (IR-V2), and Inception-v3 (I-V3).

DS	Model	Defense	DistX	BIM	PGD	C&W	D2F	SNF	TIA	KMNC	NBC	CT	NPC	LSA	HDA	HDA_o	VAE
M	L-4	Dissector	**0.97**	1.00	1.00	1.00	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	1.00	1.00	1.00
		A2D	**0.58**	1.00	1.00	1.00	0.99	0.81	0.99	0.87	0.88	0.73	0.69	0.67	1.00	1.00	1.00
	L-5	Dissector	**0.93**	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.97	0.97	0.97	0.99	1.00	1.00
		A2D	**0.68**	0.99	1.00	1.00	1.00	0.78	1.00	0.81	0.79	0.78	0.84	0.85	0.99	1.00	1.00
FM	L-4	Dissector	**0.85**	0.95	1.00	1.00	0.98	0.97	0.99	0.95	0.95	0.90	0.93	0.89	0.91	0.97	-
		A2D	**0.35**	0.91	0.99	0.98	0.87	0.74	0.77	0.80	0.80	0.60	0.63	0.46	0.95	0.97	-
	L-5	Dissector	**0.82**	0.96	0.96	0.99	0.93	0.89	9.95	0.89	0.87	0.85	0.95	0.87	0.86	0.96	-
		A2D	**0.44**	0.87	0.89	0.97	0.85	0.87	0.88	0.55	0.59	0.53	0.94	0.60	0.92	0.99	-
C	V-16	Dissector	**0.83**	0.99	0.98	0.98	0.96	0.92	0.95	0.96	0.95	0.95	0.95	0.94	0.91	0.93	-
		A2D	**0.59**	0.99	0.95	0.98	0.92	0.81	0.91	0.77	0.77	0.78	0.77	0.83	0.89	0.93	-
	R-20	Dissector	**0.89**	0.99	0.99	0.99	0.94	0.93	0.94	0.89	0.89	0.92	-	0.92	0.90	0.90	-
		A2D	**0.39**	0.96	0.95	0.78	0.93	0.95	0.83	0.56	0.56	0.89	-	0.97	0.91	0.89	-
	IR-V2	Dissector	**0.84**	0.98	0.98	0.99	0.90	0.89	0.90	0.87	0.87	0.86	-	0.86	0.91	0.90	-
		A2D	**0.24**	0.76	0.81	0.76	0.36	0.51	0.36	0.47	0.49	0.53	-	0.51	0.53	0.66	-
	I-V3	Dissector	**0.83**	0.97	0.97	0.98	0.92	0.91	0.92	0.89	0.90	0.90	-	0.89	0.93	0.93	-
		A2D	**0.27**	0.82	0.84	0.72	0.41	0.50	0.35	0.49	0.46	0.42	-	0.43	0.56	0.55	-
S	V-16	Dissector	**0.86**	0.99	0.99	0.99	1.00	0.96	0.99	0.98	0.99	0.95	0.96	0.94	0.94	0.96	0.99
		A2D	**0.36**	0.95	0.97	0.98	1.00	0.91	0.99	0.62	0.75	1.00	1.00	0.57	0.57	0.90	0.97
	R-20	Dissector	**0.88**	0.99	0.99	0.99	0.98	0.95	0.98	0.92	0.92	0.95	-	0.92	0.94	0.97	0.99
		A2D	**0.44**	0.98	0.97	0.96	0.98	0.95	0.97	0.90	0.85	1.00	-	0.98	0.65	0.91	0.99

To demonstrate the capability in enhancing robustness, we select test suites with diverse distributions (*i.e.*, *distribution difference diversity* and *target class diversity*). For each pair (c, c'), we split the distribution difference $[DF_1, DF_{30}]$ into 10 intervals, where DF_n represents the best fitness value in the n^{th} iteration. Note that the fitness values in multiple iterations may fall into the same interval. To achieve the *distribution difference diversity*, we randomly select an iteration from each interval and collect its best chromosome (*i.e.*, 10 chromosomes for each pair). To achieve the *target class diversity*, we consider all of other classes as the targets (*i.e.*, 9 targets for each source). Finally, we collect 900 test suites (10 intervals × 90 pairs) for fine-tuning in **RQ3** (*e.g.*, *Test suite 1, 2, 3, 4, . . .* in Fig. 3). To conduct a fair comparison, we collect the same number of test cases by each baseline for retraining. Specifically, for adversarial attacks, we configure different parameters such that we can generate multiple adversarial examples for each seed input. For testing tools, we first generate a large number of errors, and then randomly select the same number of inputs for retraining.

For **RQ4**, we evaluate the usefulness of *distribution difference diversity* and *target class diversity* in robustness enhancement. We collect two sets for retraining: 1) we only consider the *distribution difference diversity* and ignore the *target class diversity*. We randomly select one target class and collect multiple chromosomes from each interval, denoted as $DistXplore_{df}$ (*e.g.*, *Test suite 1, 2* in Fig. 3). 2) We select all target classes for the *target class diversity* but restrict their intervals. For each target class, we randomly select some chromosomes from only one interval, denoted as $DistXplore_t$. (*e.g.*, *Test suite 1, 3* in Fig. 3). Note that, to make a fair comparison with the results in RQ3, we control the number of test cases in $DistXplore_{df}$ and $DistXplore_t$ by collecting multiple chromosomes from an interval, such that they have the same size with the data using in RQ3 (*i.e.*, 900 test suites).

For the robustness measurement in **RQ3** and **RQ4**, we select the *empirical robustness* that is commonly used in previous works [22, 57]. The *empirical robustness* is measured by the accuracy on a validation dataset. To generate such a validation dataset, we select a new set of initial seeds (1,000) that differs from the seeds in testing. Then we run *DistXplore* and other baselines to generate errors based on new seeds. These errors found by different tools form a new test set for evaluating empirical robustness. Considering that the transformation strategies are different in different types of tools, we try to construct a balanced dataset for a fair comparison, including 9,000 errors from each type of tool, *i.e.*, adversarial attacks (3,000 for each of BIM, PGD, and C&W), distribution-unaware testing (4,500 for each configuration of DeepHunter), distribution-aware testing (3,000 for each of LSA, VAE, and HDA), and distribution-guided testing (100 for each source-target pair).

For **RQ5**, we evaluate the generalization ability of *DistXplore* by adapting it to two NLP classification tasks: *i.e.*, sentiment analysis on IMDB [36] and news classification on AG's News [67]. We fine-tune the pre-trained model BERT [8] on the two datasets, respectively.

Due to the intrinsic differences between images and textual data, we develop the text specific mutation strategies. The details about the text mutation can be found on the Website [2]. As other testing tools are mainly used in image domain, we select two NLP adversarial attacks (*i.e.*, PWWS [48] and TextFooler [23]) as the baselines. Additionally, we select the state-of-the-art method WDR [41] as the defense technique as Dissector and A2D are not suitable for BERT pre-trained models.

We follow the existing work [63] and repeat each experiment 5 times to reduce the effect of the randomness during the test generation.

4.2 Results

4.2.1 ***RQ1****:Strength of Errors.* We evaluated our method using three metrics: the *unique number of errors*, the *success rate*, and the *strength of errors*. The *unique number of errors* represents the total number of erroneous inputs generated within a given time budget. This metric is widely used in existing DL testing works [1, 3, 6, 10, 24, 34, 57, 65, 72, 73] and provides a measure of the effectiveness of

DL testing. The *success rate* measures the percentage of seed inputs from which the testing tools can generate at least one erroneous input. This metric has been employed in DL testing and adversarial attack tools [3, 11, 33, 60, 65, 66]. A higher success rate indicates that our method is capable of generating errors for a larger proportion of seed inputs. The *strength of errors* quantifies the severity or impact of the erroneous inputs generated. We emphasize the importance of generating strong errors, as weaker errors can be easily detected by existing defense tools.

Table 1 shows the results on the strength of generated errors by different methods. For *Dissector*, we use AUROC to indicate the capability on detecting errors. For *Attack-as-Defense*, we show the proportion of errors that can be detected. The symbol - in column NPC indicates that NPC cannot be used to test these DNNs since the critical paths can not be extracted. The symbol - in column VAE indicates that the VAE method does not work well on the selected task as mentioned in [9].

The overall results show that *DistXplore* (column *DistX*) can generate more strong errors that are difficult to be detected by defense techniques compared with baselines. Specifically, all errors generated by adversarial attacks underperform *DistXplore*, which may be because that they only add minor perturbations. We also found that the new advrsarial attacks outperform the classic adversarial attacks (*i.e.*, BIM, PGD, C&W). Compared with testing techniques, we can see that *DistXplore* performs better in most cases. Comparing the results between distribution-unaware testing (*i.e.*, KMNC, NBC, CT, and NPC) and distribution-aware testing (*i.e.*, HDA/HDA_o and VAE), we found that distribution-unaware testing tends to perform better because it generates some OOD data, indicating that ID errors (from distribution-aware testing) are easier to detect. *DistXplore* explicitly considers the distribution difference, which guides to generate statistically indistinguishable errors that are more difficult to detect.

Compared to other distribution-aware testing (*i.e.*, HDA/HDA_o and VAE), we found that the errors generated by LSA are harder to detect because LSA can also generate OOD data based on the surprise guidance. *DistXplore* performs better than LSA since it considers the distribution difference between each two classes and optimizes each test suite, making the discovered errors statistically indistinguishable compared with other classes.

Answers to RQ1-1: Compared with adversarial attacks and existing DL testing techniques, *DistXplore* is more effective in generating hard-to-detect errors. Existing distribution-aware testing techniques mainly focus on generating in-distribution data that could be easier to detect.

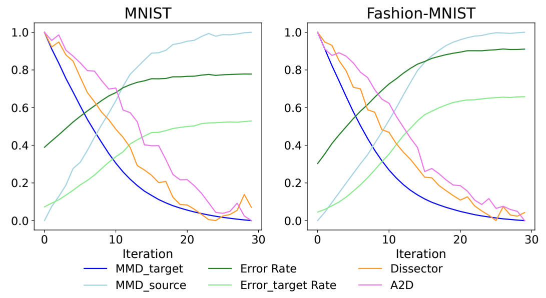

Figure 5: The average results during the optimization of *DistXplore* (model: LeNet-5)

Table 2: Results of efficiency on four datasets

DS	Mod	Metric	DistX	KMNC	NBC	LSA	HDA	VAE
M	L-4	Time (s)	**257.7**	737.9	471.4	1271.9	1937.8	1166.2
		#Error	**21,008.6**	4655.4	8029.8	8985.2	58.8	32.8
		Succ.R	**1.00**	0.65	0.96	0.96	0.59	0.33
	L-5	Time (s)	**159.4**	1260.0	758.9	2200.0	1563.7	3810.0
		#Error	**9602.4**	3414.4	6445.8	7132.4	8.8	43.4
		Succ.R	0.89	0.61	0.95	**0.96**	0.09	0.43
FM	L-4	Time (s)	**258.9**	734.1	549.0	1002.8	1963.7	-
		#Error	**21,082.8**	10,396.8	15,409.8	19,607.2	97.4	-
		Succ.R	**1.00**	0.73	0.80	0.92	0.97	-
	L-5	Time (s)	**160.8**	1810.4	1137.0	1234.0	1894.4	-
		#Error	**18,747.4**	11,064.4	15,562.8	17,756.4	94.8	-
		Succ.R	**0.99**	0.72	0.99	0.97	0.95	-
C	V-16	Time (s)	**613.9**	24,820.1	13316.1	3031.2	7255.9	-
		#Error	**26,131.4**	5924.6	8510.6	10,853.2	81.4	-
		Succ.R	**1.00**	0.79	0.94	0.93	0.82	-
	R-20	Time (s)	**605.6**	4768.5	2956.1	2931.6	6102.5	-
		#Error	**30,016.8**	8683.4	10,176.6	13,701.6	96.2	-
		Succ.R	**0.97**	0.68	0.82	0.73	0.96	-
S	V-16	Time (s)	**581.8**	24,412.3	12,488.2	6903.4	6893.6	6448.1
		#Error	**29,793.4**	2342.4	2856.2	3936.2	75.8	98.6
		Succ.R	**1.00**	0.70	0.70	0.70	0.76	0.99
	R-20	Time (s)	**606.6**	4435.4	2842.5	6645.7	6533.7	5749.1
		#Error	**29,627.4**	4122.8	5508.6	9653.6	76.8	98.6
		Succ.R	**1.00**	0.53	0.75	0.81	0.77	0.99

Fig. 5 shows the relationship between the distribution difference and the strength of errors. Due to the space limit, other results are put on our website [2]. For each pair (c, c'), we collect the best chromosome S after each iteration and calculate: 1) *MMD_target*: the distribution difference between S and the training data of target class c', 2) *MMD_source*: the distribution difference between S and the training data of source class c, 3) *Error Rate*: the proportion of errors in S, 4) *Error_target Rate*: the proportion of errors (in S) predicted as the target class and, 5) *Dissector* and *A2D*: the results detected by the different defense techniques. We average the results from all pairs, and normalize the results from 0 to 1 except for *Error Rate* and *Error_target Rate* for easier comparison.

The results show that, during the optimization, the distribution of S is getting closer to the training distribution of the target class (see *MMD_target*) and getting farther away from the source class (see *MMD_source*). Meanwhile, *Error Rate* and *Error_target Rate* are increasing, indicating that more errors are generated and gradually become statistically indistinguishable between the original class c and target class c'. The effect of indistinguishability can be further confirmed by the detection results (*i.e.*, *Dissector* and *A2D*): errors become indistinguishable and difficult to detect while the *MMD_target* decreases.

Answers to RQ1-2: The distribution difference is useful in guiding the generation of statistically indistinguishable errors, making them more difficult to detect. Compared with others, *DistXplore* generates more diverse errors.

4.2.2 ***RQ2**: Efficiency of DistXplore.*

We further study the efficiency of *DistXplore* in discovering errors, as shown in Table 2. Note that we do not set the same time to run all tools as different tools have different configuration methods. We emphasize that this paper mainly focuses on generating high-quality (*i.e.*, hard-to-detect) errors rather than merely comparing the total number of errors within a set time, as many weak errors can be easily detected by defense methods (see RQ1 results).

In Table 2, we show the time used for each tool under its configuration (Time (s)) and the total number of errors (#Error). Due to the space limit, other results are put on our website [2]. We do not show the results of adversarial attacks here because they differ from the settings of testing tools, *i.e.*, they generate an adversarial example for each seed. Overall, we can observe that *DistXplore* (column *DistX*) generates more errors while uses the shortest time. We could also observe that the existing distribution-aware testing tends to be slower due to time-consuming distribution measurements, such as the Kernel Density Estimation and VAE. Table 2 also shows the success rate of generating errors for each seed. The results show that *DistXplore* has a higher success rate than other baselines. We also notice a exception that LSA achieves higher success rate on MNIST LeNet-5. We conjecture that it is due to the optimization objective of *DistXplore* that minimizes the distribution distance, rather than specifically guiding misclassification for individual samples. In some specific datasets, the optimization may not require errors for certain seeds.

In order to evaluate the validity of the generated inputs, we conducted a manual investigation by randomly selecting 10,000 erroneous inputs from the testing outputs of each model and calculating the average validity ratio. The validity ratios were found to be 98.5%+, 96.5%+, 98.7%+, and 95.3%+ for MNIST, Fashion-MNIST, CIFAR-10, and SVHN datasets, respectively. The results demonstrate that *DistXplore* is capable of generating valid inputs with high proportions. More details are provided on our website [2].

Answers to RQ2: Compared to other DL testing tools, *DistXplore* achieves the highest efficiency in terms of the number of errors generated per second and success rates. Moreover, *DistXplore* is more effective in terms of generating valid samples.

4.2.3 ***RQ3**: Robustness Enhancement.*

For each tool, we fine-tune the original model 20 epochs following previous works [25, 38, 49] by adding the new data generated from each tool, and evaluate the empirical robustness of the new model on the validation dataset we created. Note that all data in validation dataset is predicted incorrectly by the original model. Table 3 shows the accuracy of fine-tuned models on the validation dataset. As expected, *DistXplore* outperforms the adversarial attacks, distribution-aware testing, distribution-unaware testing, and robustness-oriented testing. The overall results demonstrated the effectiveness of *DistXplore* in improving robustness. In addition, LSA achieves the second best results which outperform the results of other baselines, because LSA can generate some OOD test cases, increasing the diversity. The three modern adversarial attack techniques perform worse than the three classic techniques, because these techniques are designed for untarget attack, which decrease the distribution diversity.

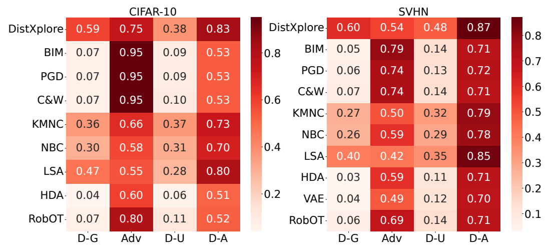

Figure 6: Accuracy on different types of dataset (model: VGG16)

Answers to RQ3-1: Overall, *DistXplore* is effective in improving robustness by generating data with different distributions. Distribution-aware testing techniques only consider ID data, making it perform poorly on errors generated by other tools.

To further interpret the results, we analyze the accuracy on different kinds of validation dataset, which is shown in Fig. 6. Other results are shown in the website [2]. Recall that our validation dataset includes 9,000 errors from distribution-guided testing *DistXplore* (*i.e.*, *D-G*), 9,000 errors from adversarial attacks (*i.e.*, *Adv*), 9,000 errors from distribution-aware testing LSA, HDA, and VAE (*i.e.*, *D-A*), and 9,000 errors from distribution-unaware testing (*i.e.*, *D-U*). Note that the dataset *D-G* and *D-U* cover more diverse transformations (*e.g.*, rotation and translation) while the dataset *Adv* and *D-A* are mainly created by the noise-based transformation. Specifically, the image transformation directly determines the distribution of the generated test cases[4] that further affects the accuracy evaluation. Taking into account that these tools use different transformations, we build such a balanced validation dataset for a fairer comparison.

Not surprisingly, each tool usually achieves better accuracy on the validation data generated by the same type of tools, because they have similar distribution, while the data from other types of tools are more likely to be OOD. For example, BIM, PGD, and C&W get much higher accuracy on *Adv* dataset since the added training data and the *Adv* data are very similar (*i.e.*, adding minor perturbation). However, the tools with only noise-based perturbation (*i.e.*, BIM, PGD, C&W, HDA, VAE, and RobOT) achieve much lower accuracy on the data *D-G* and *D-U* that use very different transformation. Their accuracy on *D-G* (<0.09) is relatively lower than that on *D-U* (>0.09), indicating some errors generated by *DistXplore* are harder to predict.

Comparing the results between DeepHunter and *DistXplore*, which use the same transformations, we found that DeepHunter achieves lower accuracy than *DistXplore* on *D-G* data because *DistXplore* generates test cases with diverse distributions, which may be OOD for DeepHunter. As for the data *D-U* generated by DeepHunter, the accuracy of *DistXplore* is slightly higher than that of DeepHunter, which indicates that the errors from *DistXplore* could cover some distribution of the data generated by DeepHunter. Considering the distribution-aware testing HDA and VAE, as they only generate ID data, they perform much worse on other dataset.

Consider the results of distribution-aware testing (HDA and VAE), adversarial attacks (BIM, PGD and C&W), and robustness-oriented testing, which use the same transformation, we found that HDA and VAE achieve lower accuracy (see Table 3), indicating that

Table 3: Results of robustness enhancement using the test cases generated by different tools on four datasets

D	M	DistXplroe	BIM	PGD	C&W	D2F	SNF	TIF	KMNC	NBC	CT	NPC	LSA	HDA	HDA_o	VAE	Robot
M	L-4	**0.81**	0.64	0.65	0.59	0.57	0.60	0.56	0.52	0.54	0.63	0.63	0.72	0.61	0.63	0.55	0.52
	L-5	**0.81**	0.66	0.65	0.66	0.41	0.49	0.46	0.65	0.68	0.45	0.45	0.76	0.59	0.57	0.57	0.56
FM	L-4	**0.73**	0.56	0.60	0.59	0.59	0.57	0.60	0.61	0.61	0.57	0.61	0.71	0.61	0.60	-	0.48
	L-5	**0.80**	0.58	0.58	0.55	0.49	0.54	0.53	0.54	0.56	0.55	0.50	0.75	0.55	0.58	-	0.41
C	V-16	**0.83**	0.53	0.53	0.53	0.30	0.33	0.30	0.73	0.70	0.36	0.36	0.80	0.51	0.55	-	0.52
	R-20	**0.76**	0.61	0.61	0.62	0.45	0.46	0.46	0.60	0.60	0.53	-	0.70	0.62	0.64	-	0.62
	IR-2	**0.97**	0.92	0.93	0.92	0.91	0.89	0.90	0.91	0.92	0.91	-	0.92	0.89	0.89	-	0.89
	I-3	**0.99**	0.93	0.93	0.93	0.92	0.91	0.92	0.92	0.93	0.93	-	0.93	0.91	0.90	-	0.89
S	V-16	**0.66**	0.55	0.54	0.54	0.22	0.28	0.27	0.55	0.57	0.29	0.30	0.59	0.50	0.50	0.49	0.52
	R-20	**0.54**	0.49	0.49	0.49	0.36	0.37	0.36	0.44	0.44	0.42	-	0.46	0.44	0.43	0.43	0.36

Table 4: Results of robustness with different distribution diversity

Dataset	Model	*DistXplore*	*DistXplore$_{df}$*	*DistXplore$_t$*
MNIST	LeNet-4	**0.81**	0.61	0.76
	LeNet-5	**0.81**	0.63	0.74
FMNIST	LeNet-4	**0.73**	0.62	0.68
	LeNet-5	**0.80**	0.65	0.72
CIFAR-10	VGG16	**0.83**	0.62	0.80
	ResNet-20	**0.76**	0.63	0.74
	IR-V2	**0.97**	0.87	0.93
	I-V3	**0.99**	0.89	0.95
SVHN	VGG16	**0.66**	0.55	0.62
	ResNet-20	**0.54**	0.42	0.47

only considering ID is less effective in improving the robustness, especially on OOD data.

The data generated by different testing tools may have different distributions, depending on their transformation and guidance strategies. All these data could be the potential inputs in the real-world deployment, and test cases generated by a tool may not cover all distributions. For example, although *DistXplore* is designed to increase the distribution diversity, it does not always cover the data distribution from other tools. In general, it can cover more unseen distributions if we gradually increase the distribution diversity.

Answers to RQ3-2: *DistXplore* can generate test cases with diverse distributions, which can identify more unseen data for further robustness improvement.

4.2.4 *RQ4: Usefulness of Distribution Diversity.*

Table 4 shows the results about the usefulness of the distribution difference diversity and target diversity. *DistXplore*, *DistXplore$_{df}$*, and *DistXplore$_t$* represents the accuracy of models fine-tuned with different data (see more configuration details in Section 4.1.4). Note that the number of data used in *DistXplore$_{df}$*, *DistXplore$_t$*, and *DistXplore* are the same. Compared to the results *DistXplore*, we found that the accuracy drops if only considering the distribution difference diversity (*DistXplore$_{df}$*) or target diversity (*DistXplore$_t$*), which indicates the usefulness of both kinds of diversity in improving the robustness.

Answers to RQ4: Both *distribution difference diversity* and *target class diversity* are useful in improving the robustness.

4.2.5 *RQ5: Generalization Ability.*

Table 5 shows the results on the strength of generated errors by different methods, *i.e.*, the percentage of errors that can be detected by existing detection methods. The overall results show that *DistXplore* can still generate strong errors than the selected baselines. Moreover, the results also demonstrate the generalizability of *DistXplore* to other domains.

Discussion on application scope. This paper primarily focuses on the classification task, which is one of the most popular and important machine learning tasks, and has been widely studied in the research area of DL testing [21, 24, 27, 32, 35, 43, 50, 52, 54, 57, 59, 63, 64]. While there is much less work on testing generation tasks in the literature due to the challenge of defining test oracles, *i.e.*, how to define the errors. Recently, researchers proposed a few metamorphic relations [17, 53] for machine translation tasks to overcome the problem. It is noteworthy that the challenge of test oracle is orthogonal to the problem we aim to solve in the paper. Considering that none of the existing works look into data distribution, we believe that *DistXplore* could also play an important role in generating test cases with better diversity for generation tasks in view that data distribution is a fundamental concept for general learning tasks.

Specifically, *DistXplore* can be extended to generation tasks by modifying the feedback of distribution differences. Currently, in classification tasks, we select other classes as targets to guide the generation of test suites for achieving diverse distributions due to the classification characteristics. For generation tasks that do not have classes, suppose there is a generation model that can generate human faces following a specific distribution (based on the training samples), we can select other datasets, such as ImageNet [7], CIFAR [28], or other image datasets, as the targets to guide the generation of test suites such that the test suites can also have diverse distributions. However, how to select the target distribution and how effectively they can help with the testing require further exploration and evaluation. We leave the extension to generation tasks as our future work.

Answers to RQ5: *DistXplore* is also useful in testing NLP models.

5 THREATS TO VALIDITY

There are some threats that could affect the validity of the results. The selected models and datasets are threats to the validity. We mitigate these threats by selecting the popular datasets and models that are used by existing DL testing works. The randomness could be a threat, which is mitigated by generating a large number of test cases over a relatively long time and running each tool 5 times in our experiments. In addition, we make our experimental

Table 5: Results of bypassing the defense techniques for NLP tasks

Dataset	Defense	DistXplore	PWWS	TextFooler
IMDB	WDR	**0.19**	0.96	0.94
AG's News	WDR	**0.22**	0.98	0.99

results publicly available. The selection of the seed inputs is a threat. We mitigate it by selecting a large number seeds (1,000) that are used by all baselines. The layer selected for calculating the MMD could be a threat to affect the results. We mitigate this problem by selecting the commonly used layer, *i.e.*, logits layer. In the future, we plan to evaluate *DistXplore* by selecting different layers and their combinations. Another threat is that the empirical robustness depends on the validation dataset, and the transformations used in selected tools are different, which could be a threat to affect the results. To mitigate this problem, we try our best to assemble a balanced validation dataset comprised of data generated from different types of testing tools (9,000 inputs generated by each type of tool). Moreover, we choose a new set of seeds to generate the validation dataset in order to avoid the overlapping between the new training dataset and validation dataset.

6 RELATED WORK

6.1 Distribution-Unaware Testing

Due to the differences between traditional software and deep neural networks, some coverage criteria have been proposed. The general idea is to define metrics for measuring the behaviors of the target DNNs while the distribution is not explicitly considered. The Neuron Coverage [46] is the first DL coverage criterion that measures the percentage of neurons activated by the given inputs. Ma *et al.*[35] then extended the Neuron Coverage and proposed a set of fine-grained coverage criteria such as k-multisection Neuron Coverage (KMNC), Neuron Boundary Coverage (NBC), and Top-k Neuron Coverage (TKNC). Although the distribution is not explicitly considered, there could be some implicit relationship between them. For example, NBC defines the covered upper and lower corner case regions, which is more related to OOD data. NPC [62] proposes two path-based coverage criteria to measure the coverage on the decision logic. A path represents a possible decision logic. Based on the coverage criteria, some automated testing techniques have been developed such as DeepXplore [46], DLFuzz [15], DeepTest [54], DeepHunter [63], DeepStellar [12], and TensorFuzz [43]

Although these techniques could also generate test cases with different distributions, none of them explicitly considers the distribution. For example, a lot of errors are generated but they may follow the similar distributions. In addition, the existing works do not consider the strength of generated errors. Differently, *DistXplore* generates strong errors that are statistically indistinguishable and enhances robustness with different distributions.

6.2 Distribution-Aware Testing

Recently, some testing works start to discuss the effect of distribution for testing, which is based on the fact that a DL model is trained on sampled training data following a specific distribution. Berend *et al.*[4] conducted an empirical study on the relationships between data distribution and existing testing techniques. They call for the attention of data-distribution awareness when designing testing methods. Zhou *et al.*[71] study the robustness of DNNs with distribution awareness. Hu *et al.*[19] study the distribution-aware seed selection methods for DNNs. Dola *et al.*[9] develop the distribution-aware testing technique that basically generates the in-distribution data by the Variational Autoencoders (VAEs). Toledo *et al.*[9] proposed the distribution-aware verification. It uses a generative model to represent the data distribution of the trained model, and then changes the original model such that all the inputs to the DNN follow the learned distribution. The most recent work [22] proposed a hierarchical distribution-aware testing method that measures both of global distribution and local distribution.

Besides, Kim *et al.*[27] propose LSA and DSA to measure the surprise adequacy (SA) of the test cases, *i.e.*, the surprise degree of a single test case compared with the training data. Although both *DistXplore* and SA consider the distance between test case(s) and training data, there are some key differences: 1) *DistXplore* measures the distribution difference between two *sets* of data while SA measures the surprise of a *single* test case In addition, considering the distance calculation, *DistXplore* is more efficient. 2) *DistXplore* is more fine-grained and considers intra-class and inter-class distribution shifts while SA mainly considers the distance between a test case and all training data. 3) The goals are not totally the same. SA is a test selection method that mainly selects surprising data. However, it is not clear whether the surprising data (from SA) is effective in generating hard-to-detect errors or enhancing model's robustness, which is our main focus. The evaluation results demonstrate that *DistXplore* is more effective.

7 CONCLUSION

In this paper, we propose a distribution-guided testing approach to evaluate and enhance DL models. To the best of our knowledge, this is the first work that explicitly generates test cases with diverse distributions. We discussed the relationship between validity and distribution, where valid out-of-distribution data is ignored by existing distribution-aware testing. We evaluated the effectiveness of *DistXplore* on 10 models and compared it with 14 state-of-the-art tools. The results demonstrate that *DistXplore* is efficient and effective in discovering hard-to-defend errors and improving robustness.

Data Availability: We provide the source code and data on: **https://github.com/l1lk/DistXplore**

ACKNOWLEDGMENTS

This research is partially supported by the National Research Foundation, Singapore, and the Cyber Security Agency under its National Cybersecurity R&D Programme (NCRP25-P04-TAICeN), the Lee Kong Chian Fellowship, the National Research Foundation Singapore and DSO National Laboratories under the AI Singapore Programme (AISG Award No: AISG2-GC-2023-008), National Key R&D Program of China (2020AAA0107702), National Natural Science Foundation of China (U21B2018, 62161160337, 61822309, U20B2049, 61773310, U1736205, 61802166) and Shaanxi Province Key Industry Innovation Program (2021ZDLGY01-02). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore and Cyber Security Agency of Singapore.

REFERENCES

[1] Zohreh Aghababaeyan, Manel Abdellatif, Mahboubeh Dadkhah, and Lionel Briand. 2023. DeepGD: A Multi-Objective Black-Box Test Selection Approach for Deep Neural Networks. *arXiv preprint arXiv:2303.04878* (2023). https://doi.org/10.48550/arXiv.2303.04878

[2] Anonymous. 2022. *DistXplore*. https://sites.google.com/view/distxplore

[3] Muhammad Hilmi Asyrofi, Zhou Yang, and David Lo. 2021. Crossasr++: A modular differential testing framework for automatic speech recognition. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1575–1579. https://doi.org/10.1145/3468264.3473124

[4] David Berend, Xiaofei Xie, Lei Ma, Lingjun Zhou, Yang Liu, Chi Xu, and Jianjun Zhao. 2020. Cats are not fish: Deep learning testing calls for out-of-distribution awareness. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. 1041–1052. https://doi.org/10.1145/3324884.3416609

[5] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*. Ieee, 39–57. https://doi.org/10.48550/arXiv.1608.04644

[6] Jaganmohan Chandrasekaran, Yu Lei, Raghu Kacker, and D Richard Kuhn. 2021. A combinatorial approach to testing deep neural network-based autonomous driving systems. In *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 57–66. https://doi.org/10.1109/ICSTW52544.2021.00022

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255. https://doi.org/10.1109/CVPR.2009.5206848

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186. https://doi.org/10.18653/V1/N19-1423

[9] Swaroopa Dola, Matthew B Dwyer, and Mary Lou Soffa. 2021. Distribution-aware testing of neural networks using generative models. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 226–237. https://doi.org/10.1109/ICSE43902.2021.00032

[10] Swaroopa Dola, Matthew B Dwyer, and Mary Lou Soffa. 2023. Input Distribution Coverage: Measuring Feature Interaction Adequacy in Neural Network Testing. *ACM Transactions on Software Engineering and Methodology* 32, 3 (2023), 1–48. https://doi.org/10.1145/3576040

[11] Yinpeng Dong, Tianyu Pang, Hang Su, and Jun Zhu. 2019. Evading defenses to transferable adversarial examples by translation-invariant attacks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4312–4321. https://doi.org/10.1109/CVPR.2019.00444

[12] Xiaoning Du, Xiaofei Xie, Yi Li, Lei Ma, Yang Liu, and Jianjun Zhao. 2019. 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*. 477–487. https://doi.org/10.1145/3338906.3338954

[13] Arthur Gretton, Karsten Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alex Smola. 2006. A kernel method for the two-sample-problem. *Advances in neural information processing systems* 19 (2006). https://doi.org/10.5555/2976456.2976521

[14] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. 2012. A kernel two-sample test. *The Journal of Machine Learning Research* 13, 1 (2012), 723–773. https://doi.org/10.5555/2188385.2188410

[15] Jianmin Guo, Yu Jiang, Yue Zhao, Quan Chen, and Jiaguang Sun. 2018. 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*. 739–743. https://doi.org/10.1145/3236024.3264835

[16] Qianyu Guo, Sen Chen, Xiaofei Xie, Lei Ma, Qiang Hu, Hongtao Liu, Yang Liu, Jianjun Zhao, and Xiaohong Li. 2019. An empirical study towards characterizing deep learning development and deployment across different frameworks and platforms. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 810–822. https://doi.org/10.1109/ASE.2019.00080

[17] Shashij Gupta, Pinjia He, Clara Meister, and Zhendong Su. 2020. Machine translation testing via pathological invariance. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 863–875. https://doi.org/10.1145/3368089.3409756

[18] Guosheng Hu, Yongxin Yang, Dong Yi, Josef Kittler, William Christmas, Stan Z Li, and Timothy Hospedales. 2015. When face recognition meets with deep learning: an evaluation of convolutional neural networks for face recognition. In *Proceedings of the IEEE international conference on computer vision workshops*. 142–150. https://doi.org/10.1109/ICCVW.2015.58

[19] Qiang Hu, Yuejun Guo, Maxime Cordy, Xiaofei Xie, Lei Ma, Mike Papadakis, and Yves Le Traon. 2022. An empirical study on data distribution-aware test selection for deep learning enhancement. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 4 (2022), 1–30. https://doi.org/10.1145/3511598

[20] Qiang Hu, Yuejun Guo, Xiaofei Xie, Maxime Cordy, Mike Papadakis, and Yves Le Traon. 2023. LaF: Labeling-Free Model Selection for Automated Deep Neural Network Reusing. *ACM Trans. Softw. Eng. Methodol.* (jul 2023). https://doi.org/10.1145/3611666 Just Accepted.

[21] Qiang Hu, Yuejun Guo, Xiaofei Xie, Maxime Cordy, Mike Papadakis, Lei Ma, and Yves Le Traon. 2023. Aries: Efficient Testing of Deep Neural Networks via Labeling-Free Accuracy Estimation. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1776–1787. https://doi.org/10.1109/ICSE48619.2023.00152

[22] Wei Huang, Xingyu Zhao, Alec Banks, Victoria Cox, and Xiaowei Huang. 2022. Hierarchical Distribution-Aware Testing of Deep Learning. *arXiv preprint arXiv:2205.08589* (2022). https://doi.org/10.1109/ICSE43902.2021.00032

[23] Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. 2020. Is bert really robust? a strong baseline for natural language attack on text classification and entailment. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 8018–8025. https://doi.org/10.1609/AAAI.V34I05.6311

[24] Haibo Jin, Ruoxi Chen, Haibin Zheng, Jinyin Chen, Yao Cheng, Yue Yu, Tieming Chen, and Xianglong Liu. 2023. Excitement surfeited turns to errors: Deep learning testing framework based on excitable neurons. *Information Sciences* 637 (2023), 118936. https://doi.org/10.1016/j.ins.2023.118936

[25] Jaeyoung Kang, Behnam Khaleghi, Tajana Rosing, and Yeseong Kim. 2022. Openhd: A gpu-powered framework for hyperdimensional computing. *IEEE Trans. Comput.* 71, 11 (2022), 2753–2765. https://doi.org/10.1109/TC.2022.3179226

[26] Yigitcan Kaya, Bilal Zafar, Sergul Aydore, Nathalie Rauschmayr, and Krishnaram Kenthapadi. 2022. Generating distributional adversarial examples to evade statistical detectors. (2022).

[27] Jinhan Kim, Robert Feldt, and Shin Yoo. 2019. Guiding deep learning system testing using surprise adequacy. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 1039–1049. https://doi.org/10.1109/ICSE.2019.00108

[28] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[29] Alexey Kurakin, Ian J Goodfellow, and Samy Bengio. 2018. Adversarial examples in the physical world. In *Artificial intelligence safety and security*. Chapman and Hall/CRC, 99–112. https://doi.org/10.48550/arXiv.1607.02533

[30] Jokin Labaien, Ekhi Zugasti, and Xabier De Carlos. 2021. DA-DGCEx: Ensuring validity of deep guided counterfactual explanations with distribution-aware autoencoder loss. *arXiv preprint arXiv:2104.09062* (2021). https://doi.org/10.48550/arXiv.2104.09062

[31] Yandong Li, Lijun Li, Liqiang Wang, Tong Zhang, and Boqing Gong. 2019. Nattack: Learning the distributions of adversarial examples for an improved black-box attack on deep neural networks. In *International Conference on Machine Learning*. PMLR, 3866–3876. https://doi.org/10.48550/arXiv.1905.00441

[32] Zhong Li, Minxue Pan, Yu Pei, Tian Zhang, Linzhang Wang, and Xuandong Li. 2022. Robust Learning of Deep Predictive Models from Noisy and Imbalanced Software Engineering Datasets. In *37th IEEE/ACM International Conference on Automated Software Engineering*. 1–13. https://doi.org/10.1145/3551349.3556941

[33] Jiadong Lin, Chuanbiao Song, Kun He, Liwei Wang, and John E Hopcroft. 2019. Nesterov accelerated gradient and scale invariance for adversarial attacks. *arXiv preprint arXiv:1908.06281* (2019). https://doi.org/10.48550/arXiv.1908.06281

[34] Yuying Liu, Pin Yang, Peng Jia, Ziheng He, and Hairu Luo. 2022. MalFuzz: Coverage-guided fuzzing on deep learning-based malware classification model. *Plos one* 17, 9 (2022), e0273804. https://doi.org/10.1371/journal.pone.0273804

[35] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, et al. 2018. Deepgauge: Multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 120–131. https://doi.org/10.1145/3238147.3238202

[36] Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*. 142–150. https://doi.org/10.5555/2002472.2002491

[37] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2017. Towards deep learning models resistant to adversarial attacks. *stat* 1050 (2017), 9. https://doi.org/10.48550/arXiv.1706.06083

[38] Arun Mallya and Svetlana Lazebnik. 2018. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 7765–7773. https://doi.org/10.1109/CVPR.2018.00810

[39] Brad L Miller, David E Goldberg, et al. 1995. Genetic algorithms, tournament selection, and the effects of noise. *Complex systems* 9, 3 (1995), 193–212. https://doi.org/10.1162/evco.1996.4.2.113

[40] AH Money, JF Affleck-Graves, ML Hart, and GDI Barr. 1982. The linear regression model: Lp norm estimation and the choice of p. *Communications in*

Statistics-Simulation and Computation 11, 1 (1982), 89–109. https://doi.org/10.1080/03610918208812247

[41] Edoardo Mosca, Shreyash Agarwal, Javier Rando-Ramirez, and Georg Groh. 2022. " That Is a Suspicious Reaction!": Interpreting Logits Variation to Detect NLP Adversarial Attacks. *arXiv preprint arXiv:2204.04636* (2022). https://doi.org/10.18653/v1/2022.acl-long.538

[42] Khan Muhammad, Amin Ullah, Jaime Lloret, Javier Del Ser, and Victor Hugo C. de Albuquerque. 2021. Deep Learning for Safe Autonomous Driving: Current Challenges and Future Directions. *IEEE Transactions on Intelligent Transportation Systems* 22, 7 (2021), 4316–4336. https://doi.org/10.1109/TITS.2020.3032227

[43] Augustus Odena and Ian J. Goodfellow. 2019. TensorFuzz: Debugging Neural Networks with Coverage-Guided Fuzzing. In *ICML*. https://doi.org/10.48550/arXiv.1807.10875

[44] Liwen Ouyang and Aaron Key. 2021. Maximum Mean Discrepancy for Generalization in the Presence of Distribution and Missingness Shift. *arXiv preprint arXiv:2111.10344* (2021). https://doi.org/10.48550/arXiv.2111.10344

[45] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroS&P)*. IEEE, 372–387. https://doi.org/10.1109/EuroSP.2016.36

[46] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Deepxplore: Automated whitebox testing of deep learning systems. In *proceedings of the 26th Symposium on Operating Systems Principles*. 1–18. https://doi.org/10.1145/3361566

[47] Adnan Qayyum, Junaid Qadir, Muhammad Bilal, and Ala Al-Fuqaha. 2021. Secure and Robust Machine Learning for Healthcare: A Survey. *IEEE Reviews in Biomedical Engineering* 14 (2021), 156–180. https://doi.org/10.1109/RBME.2020.3013489

[48] Shuhuai Ren, Yihe Deng, Kun He, and Wanxiang Che. 2019. Generating natural language adversarial examples through probability weighted word saliency. In *Proceedings of the 57th annual meeting of the association for computational linguistics*. 1085–1097. https://doi.org/10.18653/v1/P19-1103

[49] Sourjya Roy, Priyadarshini Panda, Gopalakrishnan Srinivasan, and Anand Raghunathan. 2020. Pruning filters while training for efficiently optimizing deep learning networks. In *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–7. https://doi.org/10.1109/IJCNN48605.2020.9207588

[50] Jasmine Sekhon and Cody Fleming. 2019. Towards improved testing for deep learning. In *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. IEEE, 85–88. https://doi.org/10.1109/ICSE-NIER.2019.00030

[51] Ingo Steinwart. 2001. On the influence of the kernel on the consistency of support vector machines. *Journal of machine learning research* 2, Nov (2001), 67–93. https://doi.org/10.1162/153244302760185252

[52] Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. 2018. Concolic testing for deep neural networks. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 109–119. https://doi.org/10.1145/3238147.3238172

[53] Zeyu Sun, Jie M Zhang, Yingfei Xiong, Mark Harman, Mike Papadakis, and Lu Zhang. 2022. Improving machine translation systems via isotopic replacement. In *Proceedings of the 44th International Conference on Software Engineering*. 1181–1192. https://doi.org/10.1145/3510003.3510206

[54] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering*. 303–314. https://doi.org/10.1145/3180155.3180220

[55] Felipe Toledo, David Shriver, Sebastian Elbaum, and Matthew B Dwyer. 2021. Distribution Models for Falsification and Verification of DNNs. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 317–329. https://doi.org/10.1109/ASE51524.2021.9678590

[56] Huiyan Wang, Jingwei Xu, Chang Xu, Xiaoxing Ma, and Jian Lu. 2020. Dissector: Input validation for deep learning applications by crossing-layer dissection. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 727–738. https://doi.org/10.1145/3377811.3380379

[57] Jingyi Wang, Jialuo Chen, Youcheng Sun, Xingjun Ma, Dongxia Wang, Jun Sun, and Peng Cheng. 2021. Robot: Robustness-oriented testing for deep learning systems. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 300–311. https://doi.org/10.1109/ICSE43902.2021.00038

[58] Jingyi Wang, Guoliang Dong, Jun Sun, Xinyu Wang, and Peixin Zhang. 2019. Adversarial sample detection for deep neural network through model mutation testing. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 1245–1256. https://doi.org/10.1109/ICSE.2019.00126

[59] Yan Xiao, Ivan Beschastnikh, Yun Lin, Rajdeep Singh Hundal, Xiaofei Xie, David S Rosenblum, and Jin Song Dong. 2022. Self-checking deep neural networks for anomalies and adversaries in deployment. *IEEE Transactions on Dependable and Secure Computing* (2022). https://doi.org/10.1109/TDSC.2022.3200421

[60] Cihang Xie, Zhishuai Zhang, Yuyin Zhou, Song Bai, Jianyu Wang, Zhou Ren, and Alan L Yuille. 2019. Improving transferability of adversarial examples with input diversity. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2730–2739. https://doi.org/10.1109/CVPR.2019.00284

[61] Xiaofei Xie, Hongxu Chen, Yi Li, Lei Ma, Yang Liu, and Jianjun Zhao. 2019. Coverage-guided fuzzing for feedforward neural networks. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1162–1165. https://doi.org/10.1109/ASE.2019.00127

[62] Xiaofei Xie, Tianlin Li, Jian Wang, Lei Ma, Qing Guo, Felix Juefei-Xu, and Yang Liu. 2022. NPC: N euron P ath C overage via Characterizing Decision Logic of Deep Neural Networks. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 3 (2022), 1–27. https://doi.org/10.1145/3490489

[63] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. 2019. 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*. 146–157. https://doi.org/10.1145/3293882.3330579

[64] Bing Yu, Hua Qi, Qing Guo, Felix Juefei-Xu, Xiaofei Xie, Lei Ma, and Jianjun Zhao. 2020. Deeprepair: Style-guided repairing for dnns in the real-world operational environment. *arXiv preprint arXiv:2011.09884* (2020). https://doi.org/10.1109/TR.2021.3096332

[65] Daniel Hao Xian Yuen, Andrew Yong Chen Pang, Zhou Yang, Chun Yong Chong, Mei Kuan Lim, and David Lo. 2023. ASDF: A Differential Testing Framework for Automatic Speech Recognition Systems. In *2023 IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 461–463. https://doi.org/10.1109/ICST57152.2023.00050

[66] Jianping Zhang, Jen-tse Huang, Wenxuan Wang, Yichen Li, Weibin Wu, Xiaosen Wang, Yuxin Su, and Michael R Lyu. 2023. Improving the Transferability of Adversarial Samples by Path-Augmented Method. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8173–8182. https://doi.org/10.1109/CVPR52729.2023.00790

[67] Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. *Advances in neural information processing systems* 28 (2015). https://doi.org/10.5555/2969239.2969312

[68] Zixing Zhang, Jürgen Geiger, Jouni Pohjalainen, Amr El-Desoky Mousa, Wenyu Jin, and Björn Schuller. 2018. Deep learning for environmentally robust speech recognition: An overview of recent developments. *ACM Transactions on Intelligent Systems and Technology (TIST)* 9, 5 (2018), 1–28. https://doi.org/10.1145/3178115

[69] Zhe Zhao, Guangke Chen, Jingyi Wang, Yiwei Yang, Fu Song, and Jun Sun. 2021. Attack as defense: Characterizing adversarial examples using robustness. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 42–55. https://doi.org/10.1145/3460319.3464822

[70] Tianhang Zheng, Changyou Chen, and Kui Ren. 2019. Distributionally adversarial attack. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 2253–2260. https://doi.org/10.1609/aaai.v33i01.33012253

[71] Lingjun Zhou, Bing Yu, David Berend, Xiaofei Xie, Xiaohong Li, Jianjun Zhao, and Xusheng Liu. 2020. An empirical study on robustness of DNNs with out-of-distribution awareness. In *2020 27th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 266–275. https://doi.org/10.1109/APSEC51365.2020.00035

[72] Tahereh Zohdinasab, Vincenzo Riccio, Alessio Gambi, and Paolo Tonella. 2021. Deephyperion: exploring the feature space of deep learning-based systems through illumination search. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 79–90. https://doi.org/10.1145/3460319.3464811

[73] Tahereh Zohdinasab, Vincenzo Riccio, Alessio Gambi, and Paolo Tonella. 2023. Efficient and effective feature space exploration for testing deep learning systems. *ACM Transactions on Software Engineering and Methodology* 32, 2 (2023), 1–38. https://doi.org/10.1145/3544792

Received 2023-02-02; accepted 2023-07-27