RobOT: Robustness-Oriented Testing for Deep Learning Systems

1st Jingyi Wang
Zhejiang University
wangjyee@zju.edu.cn

2nd Jialuo Chen
Zhejiang University
chenjialuo@zju.edu.cn

3rd Youcheng Sun
Queen's University Belfast
youcheng.sun@qub.ac.uk

4th Xingjun Ma
Deakin University
daniel.ma@deakin.edu.au

5th Dongxia Wang*
Zhejiang University
dxwang@zju.edu.cn

6th Jun Sun
Singapore Management University
junsun@smu.edu.sg

7th Peng Cheng
Zhejiang University
lunarheart@zju.edu.cn

2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE) | 978-1-6654-0296-5/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICSE43902.2021.00038

***Abstract*—Recently, there has been a significant growth of interest in applying software engineering techniques for the quality assurance of deep learning (DL) systems. One popular direction is deep learning testing, where adversarial examples (a.k.a. bugs) of DL systems are found either by fuzzing or guided search with the help of certain testing metrics. However, recent studies have revealed that the commonly used neuron coverage metrics by existing DL testing approaches are not correlated to model robustness. It is also not an effective measurement on the confidence of the model robustness after testing. In this work, we address this gap by proposing a novel testing framework called *Rob*ustness-*O*riented *T*esting (RobOT). A key part of RobOT is a quantitative measurement on 1) the value of each test case in improving model robustness (often via retraining), and 2) the convergence quality of the model robustness improvement. RobOT utilizes the proposed metric to automatically generate test cases valuable for improving model robustness. The proposed metric is also a strong indicator on how well robustness improvement has converged through testing. Experiments on multiple benchmark datasets confirm the effectiveness and efficiency of RobOT in improving DL model robustness, with 67.02% increase on the adversarial robustness that is 50.65% higher than the state-of-the-art work DeepGini.**

I. Introduction

Deep learning (DL) [23] has been the core driving force behind the unprecedented breakthroughs in solving many challenging real-world problems such as object recognition [38] and natural language processing [6]. Despite the success, deep learning systems are known to be vulnerable to adversarial examples (or attacks), which are slightly perturbed inputs that are imperceptibly different from normal inputs to human observers but can easily fool state-of-the-art DL systems into making incorrect decisions [5], [12], [19], [28], [48]. This not only compromises the reliability and robustness of DL systems, but also raises security concerns on their deployment in safety-critical applications such as face recognition [33], malware detection [13], medical diagnosis [10], [29] and autonomous driving [8], [25].

Noticeable efforts have been made in the software engineering community to mitigate the threats of adversarial examples and to improve the robustness of DL systems in the presence of adversarial examples [34], [36], [41]. Among them, formal verification aims to prove that no adversarial examples exist in the neighborhood of a given input. Substantial progress has been made using approaches like abstract interpretation [36], [51] and reachability analysis [40]. However, formal verification techniques are in general expensive and only scale to limited model structures and properties (e.g., local robustness [17]).

Another popular line of work is deep learning testing, which aims to generate test cases that can expose the vulnerabilities of DL models. The test cases can then be used to improve the model robustness by retraining the model, *however, this should not be taken as granted*, as recent studies have shown that test cases generated based on existing testing metrics have limited correlation to model robustness and robustness improvement after retraining [7], [15]. In this work, we highlight and tackle the problem of effectively generating test cases for improving the adversarial robustness of DL models.

There are two key elements when it comes to testing DL systems. The first element is the testing metric used to evaluate the quality of a test case or a test suite. Multiple testing metrics, including neuron coverage [34], multi-granularity neuron coverage [27] and surprise adequacy [22], have been proposed. The common idea is to explore as much diversity as possible of a certain subspace defined based on different abstraction levels, e.g., neuron activation [34], neuron activation pattern [27], neuron activation conditions [37], and neuron activation vector [22]. The second key element is the method adopted for test case generation, which is often done by manipulating a given seed input with the guidance of the testing metric. Existing test case generation techniques such as DeepXplore [34], DeepConcolic [37], DeepHunter [49] and ADAPT [24] are mostly designed to improve the neuron

* Dongxia Wang is the corresponding author.

978-1-6654-0296-5/21/$31.00 ©2021 IEEE
DOI 10.1109/ICSE43902.2021.00038

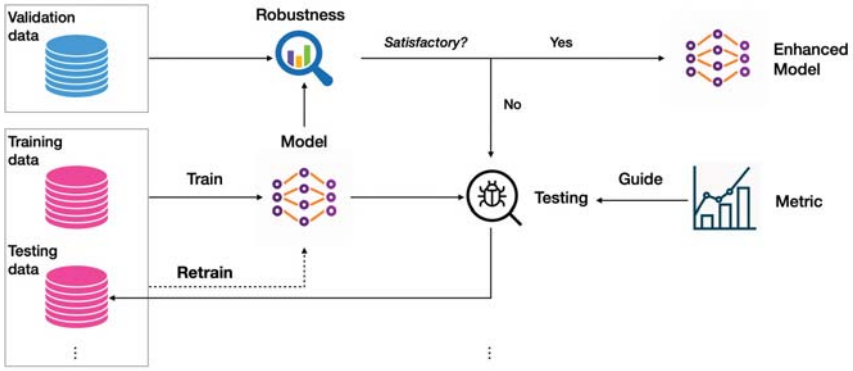

Fig. 1: Overview of RobOT testing framework.

coverage metrics of the test cases. While existing testing approaches are helpful in exposing vulnerabilities of DL systems to some extent, recent studies have found that neuron coverage metrics are not useful for improving model robustness [7], [15], [26]. As a consequence, unlike in the case of traditional program testing (where the program is surely improved after fixing bugs revealed through testing), one may not improve the robustness of the DL system after testing.

In this work, we address the above-mentioned limitations of existing DL testing approaches by proposing a novel DL testing framework called RobOT (i.e., *Rob*ustness-*O*riented *T*esting), which integrates the DL (re)training with the testing. As illustrated in Fig. 1, RobOT distinguishes itself from existing neuron coverage guided testing works in the following important aspects. First, RobOT is robustness-oriented. RobOT takes a user-defined requirement on the model robustness as input and integrates the retraining process into the testing pipeline. RobOT iteratively improves the model robustness by generating test cases based on a testing metric and retraining the model. Second, in RobOT, we propose a novel set of lightweight metrics that are strongly correlated with model robustness. The metrics can quantitatively measure the relevance of each test case for model retraining, and are designed to favor test cases that can significantly improve model robustness, which is in contrast to existing coverage metrics that have little correlation with model robustness. Furthermore, the proposed metrics can in turn provide strong evidence on the model robustness after testing. The output of RobOT is an enhanced model that satisfies the robustness requirement.

In a nutshell, we make the following contributions.

- We propose a robustness-oriented testing (RobOT) framework for DL systems. RobOT provides an end-to-end solution for improving the robustness of DL systems against adversarial examples.
- We propose a new set of lightweight testing metrics that quantify the importance of each test case with respect to the model's robustness, which are shown to be stronger indicators of the model's robustness than existing metrics.
- We implement in RobOT, a set of fuzzing strategies guided by the proposed metrics to automatically generate high-quality test cases for improving the model robustness.

RobOT is publicly available as an open-source self-contained toolkit [1]. Experiments on four benchmark datasets confirm the effectiveness of RobOT in improving model robustness. Specifically, RobOT achieves 50.65% more robustness improvement on average compared to state-of-the-art work DeepGini [9].

II. Background

A. Deep Neural Networks

In this work, we focus on deep learning models, e.g., deep neural networks (DNNs) for classification. We introduce a conceptual deep neural network (DNN) as an example in Fig. 2 for simplicity and remark that our approach is applicable for state-of-the-art DNNs in our experiments like ResNet [16], VGG [35], etc.

DNN: A DNN classifier is a function $f : X \to Y$, which maps an input $x \in X$ (often preprocessed into a vector) into a label in $y \in Y$. As shown in Fig. 2, a DNN f often contains an input layer, multiple hidden layers and an output layer. We use θ to denote the parameters of f which assigns weights to each connected edge between neurons. Given an input x, we can obtain the output of each neuron on x, i.e., $f(x, ne)$, by calculating the weighted sum of the outputs of all the neurons in its previous layer and then applying an activation function (e.g., Sigmoid, hyperbolic tangent (tanh), or rectified linear unit (relu)) ϕ. Given a dataset $D = \{(x_i, y_i)\}_{i=1}^{n}$, a DNN is often trained by solving the following optimization problem:

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \mathcal{J}(f_\theta(x_i), y_i) \tag{1}$$

, where $\mathcal{J}$ is a loss function which calculates a loss by comparing the model output $f_\theta(x_i)$ with the ground-truth

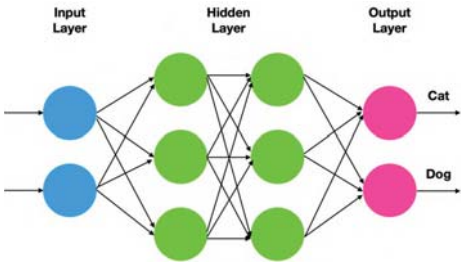

Fig. 2: An example DNN to predict cat or dog.

label y_i. The most commonly used loss function for multi-class classification tasks is the categorical cross-entropy. The DNN is then trained by computing the gradient w.r.t. the loss for each sample in D and updating θ accordingly.

B. Deep Learning Testing

Most existing deep learning testing works are based on neuron coverage [34] or its variants [27]. Simply speaking, a neuron ne is covered if there exists at least one test case x where $f(x, ne)$ is larger than a threshold and thus been activated. We omit the details of other variants and briefly introduce the following testing methods as representatives. We also provide pointers for more details.

DeepXplore [34] is the first testing work for DNN. DeepXplore proposed the first testing metric, i.e., neuron coverage and a differential testing framework to generate test cases to improve the neuron coverage.

DeepHunter [49] is a fuzzing framework which randomly selects seeds to fuzz guided by multi-granularity neuron coverage metrics defined in [27].

ADAPT [24] is another recent work which adopts multiple adaptive strategies to generate test cases which could improve the multi-granularity neuron coverage metrics defined in [27].

Adversarial Attacks Beside the above testing methods, traditional adversarial attacks like FGSM [12], JSMA [32], C&W [5] and PGD [30] attacks are also used to generate test cases in multiple works.

C. Problem definition

Unlike existing coverage guided testing works, our goal is to design a robustness-oriented testing framework to improve the DL model robustness by testing. Two key problems are to be answered: 1) how can we design testing metrics which are strongly correlated with model robustness? 2) how can we automatically generate test cases favoring the proposed testing metrics?

III. The RobOT Framework

In this section, we present RobOT, a novel robustness-oriented framework for testing and re-training DL systems. The overall framework of RobOT is shown in Figure 1. We assume that *a requirement on the model robustness* (Section III-A) is provided in prior for quality assurance purpose. Note that the requirement is likely application-specific, i.e., different applications may have different requirements on the level of robustness.

RobOT integrates the DL (re)training into the testing framework. It starts from the initial training dataset D_0, and trains an initial DNN f_0 in the standard way. Then, it applies a fuzzing algorithm (see Section III-E) which is guided by our proposed testing metrics (see Section III-C) to generate a new set of test cases D_t, for retraining the model f_0 to improve its adversarial robustness. The retraining step distinguishes RobOT from existing DL testing works and it places a specific requirement on how the test cases in D_t are generated and selected, i.e., the test cases must be helpful in improving f_0's robustness after retraining. We discuss how the test cases are generated in the rest of this section.

RobOT iteratively generates the test suite D_t and retrains the model f_n at each iteration. Afterwards, it checks whether the robustness of the new model f_n is satisfactory using an independent adversarial validation dataset D_v, subject to an acceptable degrade of the model's accuracy on normal/non-adversarial data. If the answer is yes, it terminates and outputs the final model f_n; otherwise, RobOT continues until the model robustness is satisfactory or a predefined testing budget is reached. In the following, we illustrate each component of RobOT in detail.

A. DL Robustness: A Formal Definition

Although many DL testing works in the literature claim a *potential* improvement on the DL model robustness by retraining using the test suite generated, such a conjecture is often not rigorously examined. This is partially due to the ambiguous definition of robustness. For instance, the evaluations of [34], [37], [39], [49] are based on accuracy, in particular empirical accuracy on the validation set [53], rather than robustness. In RobOT, we focus on improving the model *robustness* (without sacrificing accuracy significantly), and we begin with defining robustness.

Definition 1: ***Global Robustness (GR)*** Given an input region R, a DL model $f : R \to Y$ is (σ, ϵ)-globally-robust iff $\forall x_1, x_2 \in R, ||x_1 - x_2||_p \leq \sigma \Rightarrow \ ||f(x_1) - f(x_2)|| \leq \epsilon$. □

Global robustness is theoretically sound, and yet extremely challenging for testing or verification [20]. To mitigate the complexity, multiple attempts have been made to constrain the robustness into local input space, such as Local Robustness [17], CLEVER [46] and Lipschitz Constant [50]. These local versions of robustness are however not ideal either, i.e., they have been shown to have their own limitations [18], [20]. For instance, CLEVER relies on the extreme value theory, making it extremely costly to calculate.

In RobOT, we adopt a practical empirical definition of robustness, which has been commonly used for model robustness evaluation in the machine learning literature [4], [30], [43], [44], [52].

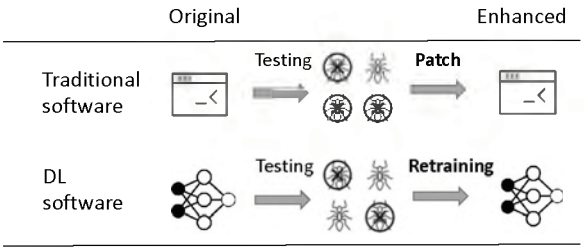

Fig. 3: Comparison between traditional and deep learning system quality assurance by testing.

Definition 2: ***Empirical Robustness (ER)*** Given a DL model $f : X \to Y$ and a validation dataset D_v, we define its empirical robustness $\mu : (f, D_v, ATT) \to [0, 1]$ as γ, where ATT denotes a given type of adversarial attack and γ is the accuracy of f on the adversarial examples obtained by conducting ATT on $\langle D_v, f \rangle$. □

Intuitively, Def. 2 evaluates a model's robustness using its accuracy on the adversarial examples crafted from a validation set D_v. Such an empirical view of DL robustness is testing-friendly and it facilitates RobOT to efficiently compare the robustness of the models before and after testing and retraining. Definition 2 is also practical, as it connects the DL robustness with many existing adversarial attacks (such as [5], [12], [30]) as a part of the definition. In particular, for the evaluation of RobOT in Section IV, we use two popular attacks, i.e., FGSM [12] and PGD (Projected Gradient Descent) [30] as ATT.

B. RobOT DL Testing: A General View

We first compare and highlight the difference between testing traditional software and deep learning systems in Fig. 3. While many testing methods (like random testing [31], symbolic execution [3], concolic testing [42] and fuzzing [11]) can be applied to identify vulnerabilities or bugs for both the traditional software and the DL systems, the workflow differs for the two after testing is done, i.e., the quality of traditional software is enhanced by patching the found bugs, whereas deep learning systems are improved via retraining. Arguably, the ultimate goal of testing is to improve the system's quality. Such improvement is guaranteed by patching bugs identified through testing in traditional software (assuming regression bugs are not frequent), i.e., the usefulness of a bug-revealing test for traditional software requires no justification. It is not obvious for DL systems, i.e., the usefulness of a test case can only be judged by taking into account the retraining step. Nevertheless, the retraining phase is largely overlooked so far in the deep learning testing literature.

Based on the Empirical Robustness definition in Def. 2, in Alg. 1, we present the high level algorithmic design of RobOT for the workflow of DL testing in Figure 3. The initial trained model f_0 is given as an input in the algorithm and the testing and retraining iterations in

Algorithm 1 RobOT(f_0, D, D_v, r, t)

1: $f = f_0$
2: **while** $ER(f, D_v, t) < r$ **do**
3: $\quad D_t \leftarrow T(f, D)$
4: $\quad D \leftarrow D \cup D_t$
5: $\quad$ Update f by retraining the model with D
6: **end while**
7: **return** f

RobOT are conducted within the main loop (Lines 2-6). The loop continues until the user-provided empirical robustness requirement is satisfied (Line 2).

RobOT aims to bridge the gap between the DL testing and retraining. Let T (Line 3) denote a fuzzing algorithm to generate test cases (guided by certain metrics). The objective of robustness-oriented testing is to improve the model robustness by testing. Formally, given a deep learning model f, the goal of RobOT at each iteration is to improve the following:

$$ER(\arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \mathcal{J}_{(x_i, y_i) \in D \cup T(f,D)}(\theta, x_i, y_i)). \quad (2)$$

Intuitively, the testing metric should be designed in such a way that after retraining with the generated test cases, the model robustness is improved. This objective directly links the testing metric to the model robustness.

In the remaining of this section, we realize the method in Line 3 by answering the question: *how should we design test metrics that are strongly correlated with the model robustness and how can we generate test cases guided by the proposed metrics?*

C. Robustness-Oriented Testing Metrics

Our goal is to design testing metrics which are strongly correlated with model robustness. We note that there have been some efforts in the machine learning community to modify the standard training procedure in order to obtain a more robust model. For instance, the most effective and successful approach so far is robust training, which incorporates an adversary in the training process so that the trained model can be robust by minimizing the loss of adversarial examples in the first place [30]:

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \max_{||x_i' - x_i||_p \leq \epsilon} \mathcal{J}(f(\theta, x_i'), y_i). \quad (3)$$

At the heart of robust training is to identify a *strong* (ideally worst-case) adversarial example x' around[1] a normal example x and train the model so that the loss on the strong adversarial example can be minimized. Robust training has shown encouraging results in training more robust models [30], [43]. This inspires us to consider deep learning testing analogously in terms of how we generate

[1] A $\epsilon-$ball defined according to a certain L_p norm.

test cases (around a normal example) and retrain the model with the test cases to improve the model robustness. The key implication is that when we design robustness-oriented testing metrics to guide testing, we should evaluate the usefulness of a test case from a loss-oriented perspective.

Let x_0 be the seed for testing. We assume that a test case x^t is generated in the neighborhood $\epsilon-$ball around x_0, i.e., $\{x|\ ||x - x_0||_p \leq \epsilon\}$ using either a testing method or an adversarial attack. The main intuition is that a test case which induces a higher loss is a stronger adversarial example, which is consequently more helpful in training robust models [30]. Based on this intuition, we propose two levels of testing metrics on top of the loss as follows.

a) Zero-Order Loss (ZOL): The first metric directly calculates the loss of a test case with respect to the DL model. Formally, given a test case x^t (generated from seed x), a DL model f, the loss of x^t on f is defined as:

$$ZOL(x^t, f) = \mathcal{J}(f(\theta, x^t), y), \tag{4}$$

where y is the ground-truth label of x. For test cases generated from the same seed, we prefer test cases with higher loss, which are more helpful in improving the model robustness via retraining.

b) First-Order Loss (FOL): The loss of generated test cases can be quite different for different seeds. In general, it is easier to generate test cases with high loss around seeds which unfortunately do not generalize well. Thus, ZOL is unable to measure the value of the test cases in a unified way. To address this problem, we propose a more fine-grained metric which could help us measure to what degree we have achieved the highest loss in the seed's neighborhood. The intuition is that, given a seed input, the loss around it often first increases and eventually converges if we follow the gradient direction to modify the seed [30]. Thus, a criteria which measures how well the loss converges can serve as the testing metric. A test case with better convergence quality corresponds to a higher loss than its neighbors. Next, we introduce First-Order Stationary Condition (FOSC) to provide a measurement on the loss convergence quality of the generated test cases.

Formally, given a seed input x_0, its neighborhood area $\mathcal{X} = \{x|\ ||x - x_0||_p \leq \epsilon\}$, and a test case x^t, the FOSC value of x^t is calculated as:

$$c(x^t) = \max_{x \in \mathcal{X}} \langle x - x^t, \nabla_x f(\theta, x^t) \rangle. \tag{5}$$

In [43], it is proved that the above problem has the following closed form solution if we take the $\infty-$norm for $\mathcal{X}$.

$$c(x^t) = \epsilon ||\nabla_x f(\theta, x^t)||_1 - \langle x^t - x_0, \nabla_x f(\theta, x^t) \rangle. \tag{6}$$

However, many existing DL testing works are generating test cases from the L_2 norm neighborhood which makes the above closed-form solution for L_∞ infeasible. We thus consider solving the formulation in Eq. 5 with L_2 norm and obtain the solution as follows:

$$c(x^t) = \epsilon ||\nabla_x f(\theta, x^t)||_2. \tag{7}$$

Proof 1: According to Cauchy–Schwarz inequality:

$$\begin{aligned} |\langle x - x^t, \nabla_x f(\theta, x^t) \rangle|^2 \leq \\ \langle x - x^t, x - x^t \rangle \cdot \langle \nabla_x f(\theta, x^t), \nabla_x f(\theta, x^t) \rangle \leq \\ \epsilon^2 \cdot (||\nabla_x f(\theta, x^t)||_2)^2 \end{aligned}$$

Since there must exist x^t such that $x - x^t$ and $\nabla_x f(\theta, x^t)$ are in the same direction, we thus have:

$$\max |\langle x - x^t, \nabla_x f(\theta, x^t) \rangle|^2 = \epsilon^2 \cdot (||\nabla_x f(\theta, x^t)||_2)^2.$$

Thus,

$$\max |\langle x - x^t, \nabla_x f(\theta, x^t) \rangle| = \epsilon \cdot ||\nabla_x f(\theta, x^t)||_2.$$

Note that FOSC (in both Eq. 6 and Eq. 7) is cheap to calculate, whose main cost is a one-time gradient computation (easy to obtain by all the DL frameworks). The FOSC value represents the first-order loss of a given test case. The loss of a test case converges and achieves the highest value if its FOSC value equals zero. Thus, a smaller FOSC value means a better convergence quality and a higher loss.

c) Comparison with Neuron Coverage Metrics: Compared to neuron coverage metrics, our proposed loss based metrics have the following main differences. First, both ZOL and FOL are strongly correlated to the adversarial strength of the generated test cases and the model robustness. Thus, our metrics can serve as strong indicators on the model's robustness after retraining. Meanwhile, our metrics are also able to measure the value of each test case in retraining, which helps us select valuable test cases from a large amount of test cases to reduce the retraining cost.

D. FOL Guided Test Case Selection

In the following, we show the usefulness of the proposed metric through an important application, i.e., test case selection from a massive amount of test cases. Note that by default, we use the FOL metric hereafter due to the limitation of ZOL as described above. Test case selection is crucial for improving the model robustness with limited retraining budget. The key of test case selection is to quantitatively measure the value of each test case. So far this problem remains an open challenge. Prior work like DeepGini has proposed to calculate a Gini index of a test case from the model's output probability distribution [9]. DeepGini's intuition is to favor those test cases with most uncertainty (e.g., a more flat distribution) under the current model's prediction. Compared to DeepGini, FOL contains fine-grained information at the loss level and is strongly correlated with model robustness.

Given a set of test cases D^t, we introduce two strategies based on FOL to select a smaller set $D^s \subset D^t$ for retraining the model as follows. Let $D^t = [x_1, x_2, \cdots, x_m]$ be a ranked list in descending order by FOL value, i.e., $FOL(x_i) \geq FOL(x_{i+1})$ for $i \in [1, m-1]$.

Algorithm 2 KM-ST(D^t, k, n)

1: $D^s = \emptyset$
2: Let max and min be the maximum and minimum FOL value respectively
3: Equally divide range $[min, max]$ into k sections $KR = [R_1, R_2, \cdots, R_k]$
4: **for** Each FOL range $r \in [R_1, R_2, \cdots, R_k]$ **do**
5: Randomly select n/k samples D^r from D^t whose FOL values are in r
6: $D^s = D^s \cup D^r$
7: **end for**
8: **return** D^s

a) K-Multisection Strategy (KM-ST): The idea of KM-ST is to uniformly sample the FOL space of D^t. Algo. 2 shows the details. Assume we need to select n test cases from D^t. We equally divide the range of FOL into k sections (KR) at line 3. Then for each range $r \in KR$, we randomly select the same number of test cases at line 5.

b) Bi-End Strategy (BE-ST): The idea of BE-ST is to form D^s by equally combining test cases with small and large FOSC values. This strategy mixes test cases of strong and weak adversarial strength, which is inspired by a recent work on improving standard robust training [21]. Given a ranked D^t, we can simply take an equal number of test cases from the two ends of the list to compose D^s.

Figure 4 shows the loss map of the selected test cases according to different strategies. We could observe that BE-ST prefers test cases of higher loss, KM-ST uniformly samples the loss space, while DeepGini often prefers test cases with lower loss.

E. FOL Guided Fuzzing

Next, we introduce a simple yet efficient fuzzing strategy to generate test cases based on FOL. Note that since we have no prior knowledge of the FOL distribution, we are not able to design fuzzing strategy for KM-ST. Instead, we design a fuzzing algorithm for the BE-ST strategy. The idea is to greedily search for test cases in two directions, i.e., with both small or large FOL values.

Algo. 3 presents the details. The inputs include the model f, the list of seeds to fuzz $seeds\_list$, the fuzzing region ϵ, the threshold on the small FOL value ξ, the number of labels to optimize k, a hyper-parameter λ on how much we favor FOL during fuzzing and lastly the maximum number of iterations to fuzz for a seed $iters$. For each seed in the list, we maintain a list of seeds s_list at line 3. After obtaining a seed x from s_list (line 5), we iteratively add perturbation on it from line 8 to line 28 in a way guided by FOL. We set the following objective for optimization (line 9).

$$obj = \sum_{i=2}^{k} P(c_i) - P(c_1) + \lambda \cdot FOL(x'), \tag{8}$$

Algorithm 3 FOL-Fuzz($f, seeds\_list, \epsilon, \xi, k, \lambda, iters$)

1: Let $fuzz\_result = \emptyset$
2: **for** $seed \in seeds\_list$ **do**
3: Maintain a list $s\_list = [seed]$
4: **while** s_list is not empty **do**
5: Obtain a seed $x = s\_list.pop()$
6: Obtain the label of the seed $c_1 = f(x)$
7: Let $x' = x$
8: **for** $iter = 0$ to $iters$ **do**
9: Set optimization objective obj using Eq. 8
10: Obtain $grads = \frac{\nabla obj}{\nabla x'}$
11: Obtain $perb = processing(grads)$
12: Let $x' = x' + perb$
13: Let $c' = f(x')$
14: Let $dis = Dist(x', x)$
15: **if** $FOL(x') \geq FOL_m$ and $dis \leq \epsilon$ **then**
16: $FOL_m = FOL(x')$
17: $s\_list.append(x')$
18: **if** $c' != c_1$ **then**
19: $fuzz\_result.append(x')$
20: **end if**
21: **end if**
22: **if** $FOL(x') < \xi$ and $dis \leq \epsilon$ **then**
23: $s\_list.append(x')$
24: **if** $c' != c_1$ **then**
25: $fuzz\_result.append(x')$
26: **end if**
27: **end if**
28: **end for**
29: **end while**
30: **end for**
31: **return** $fuzz\_result$

where c_i is the label with the i^{th} largest softmax probability of f (c_1 with the maximum), $P(c)$ is the softmax output of label c and k is a hyper-parameter. The idea is to guide perturbation towards changing the original label (i.e., generating an adversarial example) whilst increasing the FOL value. We then obtain the gradient of the objective (line 10) and calculate the perturbation based on the gradient by multiplying a learning rate and a randomized coefficient (0.5 to 1.5) to avoid duplicate perturbation (line 11). We run two kinds of checks to achieve the BE-ST strategy at line 15 and line 22 respectively. If the FOL value of the new sample after perturbation (x') is either increasing (line 15) or is smaller than a threshold (line 22), we add x' to the seed list (line 17 and line 23). Furthermore, we add x' to the fuzzing result if it satisfies the check and has a different label with the original seed x (line 19 and line 25). Note that compared to neuron coverage guided fuzzing algorithms which need to profile and update neuron coverage information [24], [49], our FOL guided fuzzing algorithm is much more lightweight, i.e., whose main cost is to calculate a gradient at each step.

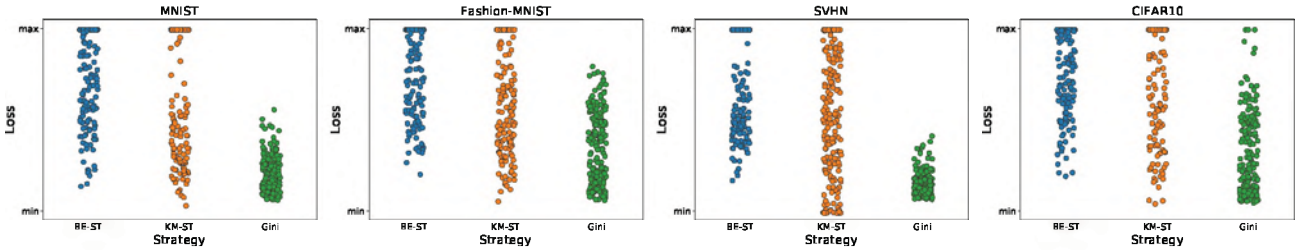

Fig. 4: Loss of selected test cases for different datasets using different strategies.

TABLE I: Datasets and models.

Dataset	Training	Testing	Model	Accuracy
MNIST	60000	10000	LeNet-5	99.02%
Fashion-MNIST	60000	10000	LeNet-5	90.70%
SVHN	73257	26032	LeNet-5	88.84%
CIFAR10	50000	10000	ResNet-20	90.39%

TABLE II: Test case generation details.

Testing Method	Parameter	MNIST	SVHN	Fashion-MNIST	CIFAR10
FGSM	Step size	0.3	0.03	0.03	0.01
PGD	Steps	10	10	10	10
	Step size	0.3/6	0.03/6	0.3/6	0.01/6
DeepXplore	Relu threshold	0.5	0.5	0.5	0.5
DLFuzz/ADAPT	Time per seed	10 s	10 s	10 s	20 s
	Relu threshold	0.5	0.5	0.5	0.5

IV. Experimental Evaluation

We have implemented RobOT as a self-contained tookit with about 4k lines of Python code. The source code and all the experiment details are available at [1]. In the following, we evaluate RobOT through multiple experiments.

A. Experiment Settings

a) Datasets and Models: We adopt four widely used image classification benchmark datasets for the evaluation. We summarize the details of the datasets and models used in Tab. I.

b) Test Case Generation: We adopt two kinds of adversarial attacks and three kinds of coverage-guided testing approaches to generate test cases for the evaluation in the following.

We summarize all the configurations of the test case generation algorithms in Tab. II.

c) Test Case Selection Baseline: We adopt the most recent work DeepGini [9] as the baseline of the test case selection strategy. DeepGini calculates a Gini index for each test case according to the output probability distribution of the model. A test case with larger Gini index is considered more valuable for improving model robustness.

d) Robustness Evaluation: We adopt Def. 2 to empirically evaluate a model's robustness. In practice, we compose a validation set of adversarial examples D_v for each dataset by combining the adversarial examples generated using both FGSM and PGD (10000 each). The attack parameters are the same with Tab. II. We then evaluate a model's robustness by calculating its accuracy on D_v.

B. Research Questions

RQ1: What is the correlation between our FOL metric and model robustness? To answer this question, we first select three models with different robustness levels for each dataset. The first model (Model 1) is the original trained model. The second model (Model 2) is a robustness-enhanced model which is retrained[2] by augmenting 5% of the generated test cases and is more robust than Model 1. The third model (Model 3) is a robustness-enhanced model which is retrained by augmenting 10% of the generated test cases and is most robust. Then, for each model, we conduct adversarial attacks to obtain a same number (10000 for FGSM and 10000 for PGD) of adversarial examples.

We show the FOL distribution of the adversarial examples for different models in Fig. 5. We observe that there is a strong correlation between the FOL distribution of adversarial examples and the model robustness. Specifically, *the adversarial examples of a more robust model have smaller FOL values*. This is clearly evidenced by Fig. 5, i.e., for every dataset, the probability density is intensively distributed around zero for Model 3 (the most robust model) while is steadily expanding to larger FOL values for Model 2 and Model 1 (with Model 1 larger than Model 2). The underlying reason is that a more robust model in general has a more *flat* loss distribution and thus a smaller FOL value (since it is based on the loss gradient).

In addition, we also observe that adversarial examples crafted by stronger attacks have smaller FOL values. Fig. 6 shows the FOL distribution of adversarial examples from attacking the CIFAR10 model with FGSM and PGD respectively. We could observe that adversarial examples from PGD have significantly smaller FOL values than FGSM. The reason is that stronger attacks like PGD are generating adversarial examples that have better loss convergence quality and induce higher loss.

We thus have the following answer to RQ1:

[2]Retaining in this work takes 10 (40 for CIFAR10) additional epochs based on the original model.

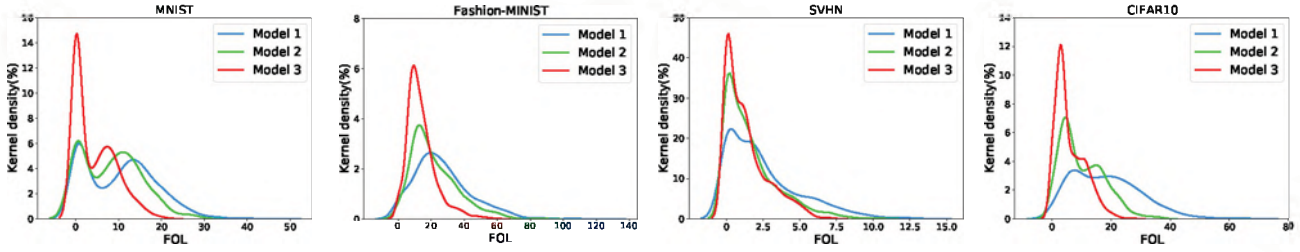

Fig. 5: FOL distribution of adversarial examples for models with different robustness.

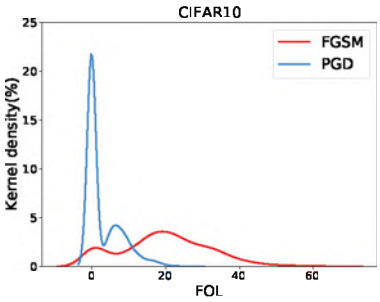

Fig. 6: FOL distribution of adversarial examples from FGSM and PGD for CIFAR10 model.

Answer to RQ1: FOL is strongly correlated with model robustness. A more robust model have smaller FOL values for adversarial examples.

RQ2: How effective is our FOL metric for test case selection? To answer the question, we first generate a large set of test cases using different methods, and then adopt different test case selection strategies (i.e., BE-ST, KM-ST and DeepGini) to select a subset of test cases with the same size to retrain the model. A selection strategy is considered more effective if the retrained model with the selected test cases is more robust.

We distinguish two different kinds of test case generation algorithms which are both used in the literature, i.e., adversarial attacks and neuron coverage-guided algorithms, for more fine-grained analysis. For adversarial attacks, we adopt FGSM (weak) and PGD (strong) attacks to generate a combined set of test cases. For DeepXplore, DLFuzz and ADAPT, we generate a set of test cases for each of them. The parameters used are consistent with Tab. II. For each set of test cases, we use BE-ST, KM-ST and DeepGini strategy respectively to select x (ranging from 1 to 10) percent of them to obtain a retrained model and evaluate its model robustness.

Fig. 7 shows the results. We observe that for all the strategies, the retrained model obtained improved resilience to adversarial examples to some extent. Besides, the model's robustness steadily improves as we augment more test cases (from 1% to 10%) for retraining. However, we could also observe that in almost all cases (except 1 case), our FOL guided strategies (both BE-ST and KM-ST) have significantly better performance than DeepGini, i.e., achieving 30.48%, 84.62%, 54.91% and 35.92% more robustness improvement on average for the four different sets of test cases. The reason is that FOL is able to select test cases which have higher and more diverse loss than DeepGini (as shown in Fig. 4 previously), which are better correlated with model robustness. Meanwhile, we observe that the retrained models maintain high accuracy on the test set as well (as summarized in Tab. III).

Besides, we observe that different test case generation algorithms obtain different robustness improvements. Among DeepXplore, DLFuzz and ADAPT, ADAPT and DLFuzz have the highest (53.39% on average) and lowest (31.18% on average) robustness improvement respectively while DeepXplore is in the middle (48.36% on average). Adversarial attacks often achieve higher robustness improvement than all three neuron coverage-guided fuzzing algorithms for simpler datasets such as MNIST, Fashion-MNIST and SVHN. This casts shadow on the usefulness of the test cases generated by neuron coverage-guided fuzzing algorithms in improving model robustness and is consistent with [7], [15], [26].

We further conduct experiments to evaluate and compare how robust the retrained models are when using adversarial examples generated in different ways: one from the attacks (Fig. 7), and the other from different testing algorithms. We summarize the result in Tab. IV. We observe that the robustness drops noticeably (which is especially the case for CIFAR10), i.e., 18.64%, 26.41% and 23.09% for DeepXplore, DLFuzz, and ADAPT each on average (compared to the results in Fig. 7). Nevertheless, our test case selection strategies still outperform DeepGini in all cases. This shows that adversarial examples from adversarial attacks alone are insufficient. It is necessary to improve the diversity of test cases for retraining from a perspective that is well correlated with model robustness.

We thus have the following answer to RQ2:

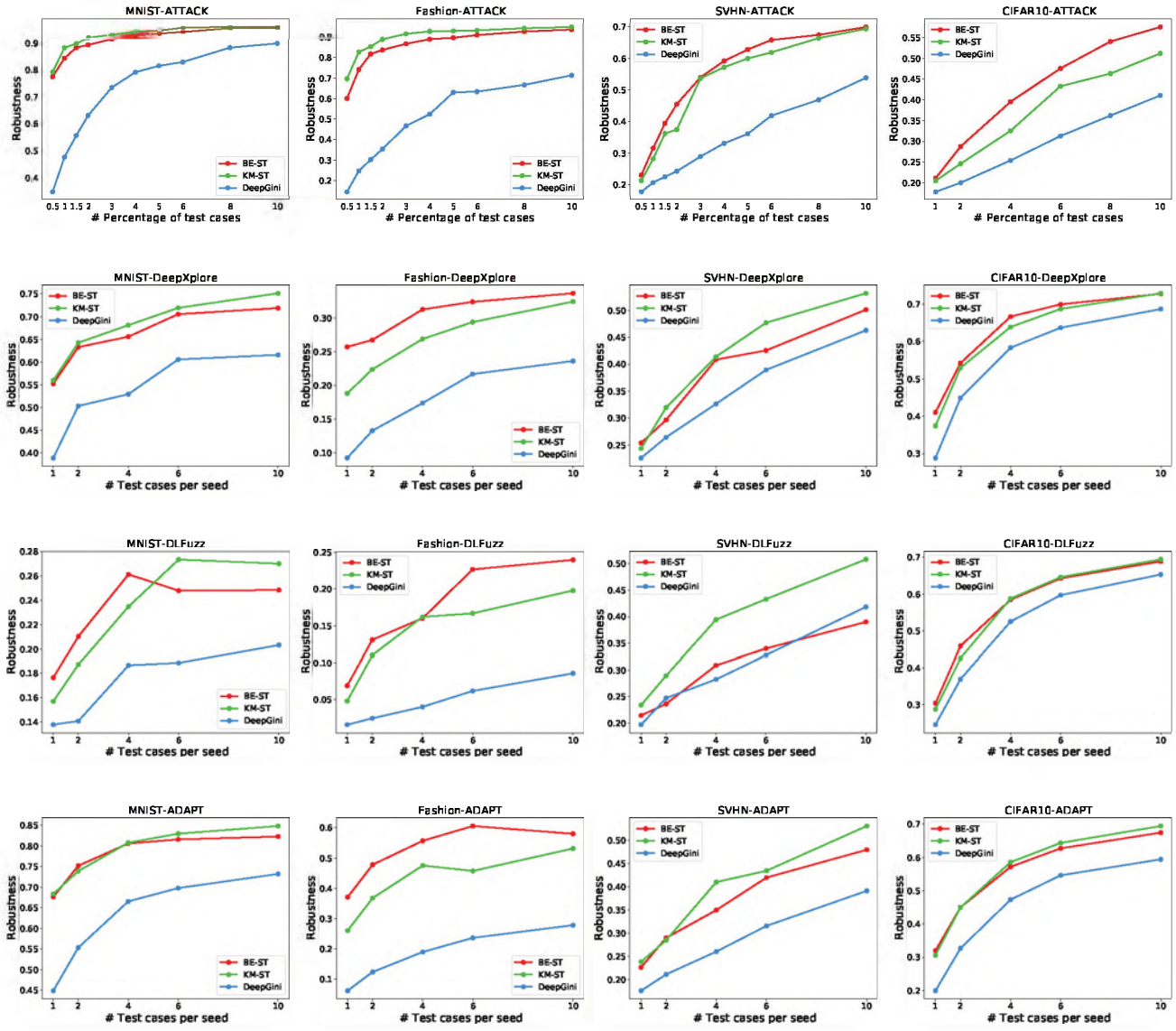

Fig. 7: Test case selection and robustness improvement with different strategies.

TABLE III: Test accuracy of model before and after retraining with 10 percent of generated test cases using adversarial attacks.

Dataset	Original	Retrained
MNIST	99.02%	98.95%
Fashion-MNIST	90.70%	90.63%
SVHN	88.84%	87.13%
CIFAR10	90.39%	90.13%

Answer to RQ2: FOL guided test case selection is able to select more valuable test cases to improve the model robustness by retraining.

RQ3: How effective and efficient is our FOL guided fuzzing algorithm? To answer the question, we compare our FOL guided fuzzing algorithm (FOL-Fuzz) with state-of-the-art neuron coverage-guided fuzzing algorithm ADAPT as follows. We run FOL-fuzz and ADAPT for a same period of time, (i.e., 5 minutes, 10 minutes and 20 minutes) to generate test cases. Then we retrain the model with the test cases to compare their robustness improvement. The hyper-parameters for FOL-Fuzz are set as follows: $\xi = 10^{-18}$, $k = 5$, $\lambda = 1$, $iters = 3$, $learning\_rate = 0.1$. The parameters for ADAPT are consistent with Tab. II.

Tab. V shows the results. We could observe that within

TABLE IV: Robustness performance of models (retrained using adversarial examples from attack algorithms) against test cases generated by DL testing tools.

	DeepXplore				DLFuzz				ADAPT			
Dataset	BE-ST	KM-ST	DeepGini	Average	BE-ST	KM-ST	DeepGini	Average	BE-ST	KM-ST	DeepGini	Average
MNIST	86.12%	80.56%	73.74%	**80.14%**	76.39%	74.73%	65.59%	**72.24%**	82.60%	75.68%	70.36%	**76.21%**
Fashion-MNIST	51.57%	47.97%	34.14%	**44.56%**	38.15%	35.44%	27.16%	**33.58%**	50.55%	47.50%	31.92%	**43.32%**
SVHN	37.10%	38.29%	27.26%	**34.55%**	32.83%	34.83%	25.34%	**31.00%**	25.71%	28.51%	19.15%	**24.46%**
CIFAR10	25.25%	20.16%	12.92%	**19.44%**	18.28%	14.20%	9.31%	**13.93%**	22.37%	18.48%	12.08%	**17.64%**
Average	**50.01%**	**46.75%**	**37.01%**		**41.41%**	**39.8%**	**31.85%**		**45.31%**	**42.54%**	**33.36%**	

TABLE V: Comparison of FOL-fuzz and ADAPT. a/b: a is the result of FOL-fuzz and b is the result of ADAPT.

	5 min		10 min		20 min	
Dataset	# Test case	Robustness↑	# Test case	Robustness↑	# Test case	Robustness↑
MNIST	1692/2125	33.62%/18.73%	3472/4521	48.04%/36.46%	7226/8943	68.02%/54.38%
Fashion-MNIST	4294/5485	40.75%/6.74%	8906/10433	53.88%/14.94%	18527/21872	69.03%/27.24%
SVHN	6236/8401	24.25%/21.3%	12465/17429	30.42%/27.52%	24864/33692	39.99%/34.51%
CIFAR10	1029/1911	18.62%/17.03%	2006/3722	22.07%/18.12%	4050/6947	27.36%/20.54%
Average	**3313/4480**	**29.31%/15.95%**	**6712/9026**	**38.6%/24.26%**	**13667/17864**	**51.1%/34.17%**

the same time limit, ADAPT generates slightly more adversarial examples, i.e., 10457 compared to 7897 of FOL-Fuzz. A closer look reveals that ADAPT tends to generate a lot of test cases around a seed towards improving the neuron coverage metrics. However, not all these tests are meaningful to improve model robustness. On the other hand, FOL-Fuzz is able to discover more valuable test cases. We could observe that using FOL-Fuzzed test cases (although less than ADAPT) to retrain the model significantly improves the model's robustness than ADAPT, i.e., 39.67% compared to 24.79% of ADAPT on average.

We thus have the following answer to RQ3:

Answer to RQ3: FOL-Fuzz is able to efficiently generate more valuable test cases to improve the model robustness.

C. Threats to Validity

First, our experiments are based on a limited set of test subjects in terms of datasets, types of adversarial attacks and neuron coverage-guided test case generation algorithms. Although we included strong adversarial attack like PGD and state-of-the-art coverage-guided generation algorithm ADAPT, it might be interesting to investigate other attacks like C&W [5] and JSMA [32], and fuzzing algorithms like DeepHunter [49]. Second, we adopt an empirical approach to evaluate the model robustness which might be different with different kinds of attacks used. So far it is still an open problem as for how to efficiently measure the robustness of DL models. We do not use more rigorous robustness metric like CLEVER [46] because it is input-specific and has high cost to calculate (e.g., hours for one input). Third, our testing framework requires a robustness requirement as input which could be application-specific and is relevant to the model as well. In practice, users could adjust the requirement dynamically.

V. Related works

This work is mainly related to the following lines of works on building more robust deep learning systems.

a) Deep Learning Testing: Extensive DL testing works are focused on designing testing metrics to expose the vulnerabilities of DL systems including neuron coverage [34], multi-granularity neuron coverage [27], neuron activation conditions [37] and surprise adequacy [22]. Along with the testing metrics, many test case generation algorithms are also proposed including gradient-guided perturbation [34], [54], black-box [47] and metric-guided fuzzing [14], [24], [49]. However, these testing works lack rigorous evaluation on their usefulness in improving the model robustness (although most of them claim so) and have been shown to be ineffective in multiple recent works [7], [15], [26]. Multiple metrics have been proposed in the machine learning community to quantify the robustness of DL models as well [2], [45], [46], [50]. However, most of them are used to evaluate local robustness and hard to calculate. Thus these metrics are not suitable to test directly. Our work bridges the gap by proposing the FOL metric which is strongly correlated with model robustness and integrate retraining into the testing pipeline for better quality assurance.

b) Adversarial Training: The key idea of adversarial training is to improve the robustness of the DL models by considering adversarial examples in the training phase. There are plenty of works on conducting adversarial attacks on DL models (of which we are not able to cover all) to generate adversarial examples such as FGSM [12], PGD [30] and C&W [5]. Adversarial training in general may overfit to the specific kinds of attacks which generate the adversarial examples for training [30] and thus can not guarantee robustness on new kinds of attacks. Later, robust training [30] is proposed to train robust models by solving a saddle point problem described in Sec. III. DL testing complements these works by generating more diverse adversarial examples.

VI. Conclusion

In this work, we propose a novel robustness-oriented testing framework RobOT for deep learning systems towards improving model robustness against adversarial examples. The core of RobOT is a metric called FOL to quantify both the value of each test case in improving model robustness (often via retraining) and the convergence quality of the model robustness improvement. We also propose to utilize the proposed metric to automatically fuzz for more valuable test cases to improve model robustness. We implemented RobOT as a self-contained open-source toolkit. Our experiments on multiple benchmark datasets verify the effectiveness and efficiency of RobOT in improving DL model robustness, i.e., with 67.02% increasement on the adversarial robustness that is 50.65% higher than the state-of-the-art work DeepGini.

Acknowledgments

This work was supported by the National Key R&D Program of China (Grant No. 2020YFB2010901). This work was also supported by the NSFC Program (Grant No. 62061130220, 61833015 and 62088101), the Guangdong Science and Technology Department (Grant No. 2018B010107004) and the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No.: AISG-RP-2019-012).

References

[1] https://github.com/SmallkeyChen/RobOT.

[2] Wieland Brendel, Jonas Rauber, Matthias Kümmerer, Ivan Ustyuzhaninov, and Matthias Bethge. Accurate, reliable and fast robustness evaluation. In *Advances in Neural Information Processing Systems*, pages 12861–12871, 2019.

[3] Cristian Cadar, Daniel Dunbar, Dawson R Engler, et al. Klee: unassisted and automatic generation of high-coverage tests for complex systems programs. In *OSDI*, volume 8, pages 209–224, 2008.

[4] Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian Goodfellow, Aleksander Madry, and Alexey Kurakin. On evaluating adversarial robustness. *arXiv preprint arXiv:1902.06705*, 2019.

[5] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. IEEE, 2017.

[6] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(ARTICLE):2493–2537, 2011.

[7] Yizhen Dong, Peixin Zhang, Jingyi Wang, Shuang Liu, Jun Sun, Jianye Hao, Xinyu Wang, Li Wang, Jin Song Dong, and Dai Ting. There is limited correlation between coverage and robustness for deep neural networks. *arXiv preprint arXiv:1911.05904*, 2019.

[8] Ranjie Duan, Xingjun Ma, Yisen Wang, James Bailey, A Kai Qin, and Yun Yang. Adversarial camouflage: Hiding physical-world attacks with natural styles. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1000–1008, 2020.

[9] Yang Feng, Qingkai Shi, Xinyu Gao, Jun Wan, Chunrong Fang, and Zhenyu Chen. 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*, ISSTA 2020, page 177–188, New York, NY, USA, 2020. Association for Computing Machinery.

[10] Samuel G Finlayson, John D Bowers, Joichi Ito, Jonathan L Zittrain, Andrew L Beam, and Isaac S Kohane. Adversarial attacks on medical machine learning. *Science*, 363(6433):1287–1289, 2019.

[11] Patrice Godefroid, Adam Kiezun, and Michael Y Levin. Grammar-based whitebox fuzzing. In *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 206–215, 2008.

[12] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

[13] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. Adversarial examples for malware detection. In *European Symposium on Research in Computer Security*, pages 62–79. Springer, 2017.

[14] Jianmin Guo, Yu Jiang, Yue Zhao, Quan Chen, and Jiaguang Sun. 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*, pages 739–743, 2018.

[15] Fabrice Harel-Canada, Lingxiao Wang, Muhammad Ali Gulzar, Quanquan Gu, and Miryung Kim. Is neuron coverage a meaningful measure for testing deep neural networks? In *Proceedings of the Joint Meeting on Foundations of Software Engineering (FSE)*, 2020.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[17] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification*, pages 3–29. Springer, 2017.

[18] Todd Huster, Cho-Yu Jason Chiang, and Ritu Chadha. Limitations of the lipschitz constant as a defense against adversarial examples. In Carlos Alzate, Anna Monreale, Haytham Assem, Albert Bifet, Teodora Sandra Buda, Bora Caglayan, Brett Drury, Eva García-Martín, Ricard Gavaldà, Irena Koprinska, Stefan Kramer, Niklas Lavesson, Michael Madden, Ian Molloy, Maria-Irina Nicolae, and Mathieu Sinn, editors, *ECML PKDD 2018 Workshops*, pages 16–29, Cham, 2019. Springer International Publishing.

[19] Linxi Jiang, Xingjun Ma, Shaoxiang Chen, James Bailey, and Yu-Gang Jiang. Black-box adversarial attacks on video recognition models. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 864–872, 2019.

[20] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Towards proving the adversarial robustness of deep neural networks. *arXiv preprint arXiv:1709.02802*, 2017.

[21] Marc Khoury and Dylan Hadfield-Menell. Adversarial training with voronoi constraints. *arXiv preprint arXiv:1905.01019*, 2019.

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

[23] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[24] Seokhyun Lee, Sooyoung Cha, Dain Lee, and Hakjoo Oh. 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*, pages 165–176, 2020.

[25] Jesse Levinson, Jake Askeland, Jan Becker, Jennifer Dolson, David Held, Soeren Kammel, J Zico Kolter, Dirk Langer, Oliver Pink, Vaughan Pratt, et al. Towards fully autonomous driving: Systems and algorithms. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, pages 163–168. IEEE, 2011.

[26] Zenan Li, Xiaoxing Ma, Chang Xu, and Chun Cao. Structural coverage criteria for neural networks could be misleading. In *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, pages 89–92. IEEE, 2019.

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

[28] Xingjun Ma, Bo Li, Yisen Wang, Sarah M Erfani, Sudanthi Wijewickrema, Grant Schoenebeck, Dawn Song, Michael E Houle, and James Bailey. Characterizing adversarial subspaces using local intrinsic dimensionality. *arXiv preprint arXiv:1801.02613*, 2018.

[29] Xingjun Ma, Yuhao Niu, Lin Gu, Yisen Wang, Yitian Zhao, James Bailey, and Feng Lu. Understanding adversarial attacks on deep learning based medical image analysis systems. *Pattern Recognition*, page 107332, 2020.

[30] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

[31] Carlos Pacheco and Michael D Ernst. Randoop: feedback-directed random testing for java. In *Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion*, pages 815–816, 2007.

[32] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pages 372–387. IEEE, 2016.

[33] Omkar M Parkhi, Andrea Vedaldi, and Andrew Zisserman. Deep face recognition. 2015.

[34] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 1–18. ACM, 2017.

[35] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[36] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–30, 2019.

[37] Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. Concolic testing for deep neural networks. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE 2018, page 109–119, New York, NY, USA, 2018. Association for Computing Machinery.

[38] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

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

[40] Hoang-Dung Tran, Diago Manzanas Lopez, Patrick Musau, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T Johnson. Star-based reachability analysis of deep neural networks. In *International Symposium on Formal Methods*, pages 670–686. Springer, 2019.

[41] Jingyi Wang, Guoliang Dong, Jun Sun, Xinyu Wang, and Peixin Zhang. Adversarial sample detection for deep neural network through model mutation testing. In *Proceedings of the 41st International Conference on Software Engineering*, pages 1245–1256. IEEE Press, 2019.

[42] Xinyu Wang, Jun Sun, Zhenbang Chen, Peixin Zhang, Jingyi Wang, and Yun Lin. Towards optimal concolic testing. In *Proceedings of the 40th International Conference on Software Engineering*, pages 291–302, 2018.

[43] Yisen Wang, Xingjun Ma, James Bailey, Jinfeng Yi, Bowen Zhou, and Quanquan Gu. On the convergence and robustness of adversarial training. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6586–6595, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[44] Yisen Wang, Difan Zou, Jinfeng Yi, James Bailey, Xingjun Ma, and Quanquan Gu. Improving adversarial robustness requires revisiting misclassified examples. In *International Conference on Learning Representations*, 2019.

[45] Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Duane Boning, Inderjit S Dhillon, and Luca Daniel. Towards fast computation of certified robustness for relu networks. *arXiv preprint arXiv:1804.09699*, 2018.

[46] Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, Dong Su, Yupeng Gao, Cho-Jui Hsieh, and Luca Daniel. Evaluating the robustness of neural networks: An extreme value theory approach. In *International Conference on Learning Representations*, 2018.

[47] Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. Feature-guided black-box safety testing of deep neural networks. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 408–426. Springer, 2018.

[48] Dongxian Wu, Yisen Wang, Shu-Tao Xia, James Bailey, and Xingjun Ma. Skip connections matter: On the transferability of adversarial examples generated with resnets. *arXiv preprint arXiv:2002.05990*, 2020.

[49] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. 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*, ISSTA 2019, page 146–157, New York, NY, USA, 2019. Association for Computing Machinery.

[50] Huan Xu and Shie Mannor. Robustness and generalization. *Machine learning*, 86(3):391–423, 2012.

[51] Pengfei Yang, Renjue Li, Jianlin Li, Cheng-Chao Huang, Jingyi Wang, Jun Sun, Bai Xue, and Lijun Zhang. Improving neural network verification through spurious region guided refinement. *arXiv preprint arXiv:2010.07722*, 2020.

[52] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric Xing, Laurent El Ghaoui, and Michael Jordan. Theoretically principled trade-off between robustness and accuracy. In *International Conference on Machine Learning*, pages 7472–7482. PMLR, 2019.

[53] Jie M Zhang, Mark Harman, Lei Ma, and Yang Liu. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering*, 2020.

[54] Peixin Zhang, Jingyi Wang, Jun Sun, Guoliang Dong, Xinyu Wang, Xingen Wang, Jin Song Dong, and Ting Dai. White-box fairness testing through adversarial sampling. *Proceedings of the 42th International Conference on Software Engineering (ICSE 2020), Seoul, South Korea*, 2020.