Information Sciences 614 (2022) 497–517

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Mixed and constrained input mutation for effective fuzzing of deep learning systems

Leo Hyun Park [a], Jaeuk Kim [a], Jaewoo Park [a], Taekyoung Kwon [a,*]

[a] Information Security Lab, Graduate School of Information, Yonsei University, Seoul 03722, Republic of Korea

ARTICLE INFO

Article history:
Received 6 April 2022
Received in revised form 23 August 2022
Accepted 15 October 2022
Available online 21 October 2022

Keywords:
Coverage-guided deep learning fuzzing
Mixed and constrained mutation
Adversarial example
Image transformation algorithm

ABSTRACT

Adversarial examples cause misclassifications of deep learning (DL) systems. It isn't easy to debug misclassifications due to the intrinsic complexity of DL architecture. Thus, applying coverage-guided fuzzing, a widely used technique to find crashes in complex software, is promising. However, mutation strategies of DL fuzzers, such as DeepHunter, have a limitation. They restrict multiple transformations and so penalize generating diverse inputs. Otherwise, they cause significant distortion, rendering invalid inputs, such as unrecognizable or ambiguous to humans. However, multiple transformations are critical in mutation-based fuzzing. To address this problem, we propose the mixed and constrained mutation (MCM) for DL fuzzers. Human perception-based constraints of MCM avoid significant distortion in a single transformation and the aggregation of multiple transformations. We verify transformation parameters through a survey with 15 participants on each MNIST, STL-10, and ImageNet dataset to implement such constraints, followed by statistical tests. MCM returns valid inputs in almost every fuzzing iteration. Furthermore, MCM improved the fuzzing performance on various DL architectures on MNIST, STL-10, and ImageNet compared to DeepHunter: MCM discovered 17.6% more seeds showing new coverage and 132% more adversarial examples on average. These adversarial examples correspond to more than double the incorrect classes for each original image than DeepHunter.

1. Introduction

Deep learning has enabled significant progress in various domains such as visual recognition [1,2] and natural language processing [3]. However, even with high-accuracy test datasets, deep learning models are still unable to cope with corner-case inputs that are rarely found, but if found, ambiguous to the models (e.g., a collision of self-driving vehicles because of the incorrect steering angle). For instance, adversarial examples are a serious threat to deep learning models. An adversarial example is the compound of a clean input and imperceptible perturbation that is created from transformation algorithms or the gradient of the loss function. Humans recognize adversarial examples as legitimate, but deep learning models classify them into incorrect classes. In security-critical domains such as medical diagnosis [4], self-driving vehicles [5], and anomaly detection [6], a few malfunctions can lead to fatal accidents. This hinders the application of deep learning-based systems in the real world. Therefore, deep learning models need to guarantee not only the accuracy of the validation set but also the robustness of corner-case inputs.

* Corresponding author.
E-mail addresses: dofi@yonsei.ac.kr (L.H. Park), freak0wk@yonsei.ac.kr (J. Kim), jaewoo1218@yonsei.ac.kr (J. Park), taekyoung@yonsei.ac.kr (T. Kwon).

https://doi.org/10.1016/j.ins.2022.10.079
0020-0255/© 2022 Elsevier Inc. All rights reserved.

Thus, it is necessary to verify the prediction stability of a deep learning model for corner-case inputs. While the large-scale parameter space is an advantage of deep learning models, this property also makes such models noninterpretable, preventing debugging. A possible strategy to overcome this is to apply a testing method for traditional software to the deep learning model. The robustness of the model can be improved by feeding various inputs to the model, distinguishing corner-case inputs using the model output and designing defense methods for distinguished inputs. Coverage-guided fuzzing [7,8] is an effective automatic testing method for detecting vulnerabilities in traditional software. The mutation strategy of a fuzzer generates new inputs, and the coverage metric guides the fuzzer in executing the uncovered logic of the program.

Fuzzers in traditional software investigate which parts of the code, such as branches and basic blocks, have been executed using code coverage. However, because the logic of a deep learning model is defined by training data instead of a programmer, applying code coverage to the model is challenging. Furthermore, the meaning of execution is unclear for deep learning, because all neurons in the model always yield an output value. The coverage criteria for deep learning solved this issue by considering the difference in logic between traditional software and the deep learning model and focusing on neuron activation values [9,10].

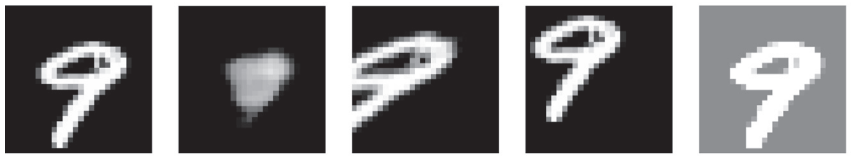
The approach that expands neuron activation coverage facilitates white-box testing of deep learning systems [11–14]. However, another challenge remains in the coverage-guided fuzzing of deep learning systems involved in creating mutated inputs: the trade-off between input diversity and distortion must be resolved. A malfunction of a deep learning model generally refers to a disagreement in the decision made by the model and an oracle (i.e., humans). For deep learning models, mutated inputs should satisfy the constraint that preserves the semantics of the original inputs. Previous studies on testing image recognition models, which is our focus in this work, used various image transformation algorithms such as rotation and adjusting brightness [12,11] to change the appearance of an input. However, if multiple transformation algorithms are combined indiscreetly, the mutation strategy generates invalid inputs that are either unrecognizable to humans (such as black or white images) or ambiguous to classify by humans (see Fig. 2 (b) and (c)). For instance, a mixture of transformation algorithms was permitted in DeepTest [12], but there was no constraint that means the condition that the transformed input is recognized as the same as the original input. Consequently, the mutation overlaps without limit, and the distortion of mutated inputs increases enormously.

Multiple transformation algorithms need to be combined to create as many inputs as possible, but the precise constraint should be accompanied to reflect real-world changes and preserve the semantics of the inputs. The constraint using the l_p-norm, which is commonly used in adversarial attacks, is inadequate for image transformation algorithms because image transformations result in a high l_p-norm, but the semantics of the original inputs remain preserved. Image transformations change the position of all pixels or add large values for all pixels. Existing fuzzers did not take this property into account while defining the constraints. DeepHunter [11] restricts the resulting image by permitting only one transformation algorithm on the original input, except for random noise. This results in a considerable reduction in the number of generable inputs.

Furthermore, the transformation parameter ranges of existing fuzzers, including DeepTest and DeepHunter, do not accurately represent real-world changes. This is because they independently defined the transformation parameters. As we can see from Fig. 2(b), invalid inputs can be generated when the parameter ranges are wider than the acceptable ranges in the case of humans. However, the mutation strategy cannot generate diverse inputs when the parameter ranges are reduced compared to the acceptable ranges of humans. In this case, as in Fig. 2(d), a mutated input with a large parameter may be rejected by the fuzzer even though the parameter is valid for humans. As both types of parameter ranges cause problems for input mutations, parameter ranges that reflect human perception should be defined.

To solve the aforementioned problems, in this study, we propose a new mutation approach for coverage-guided fuzzing of deep learning systems. Our primary objective is to generate diverse mutation results using a combination of multiple image transformation algorithms while preserving the semantics of the original inputs. To achieve this, we designed a strategy called *mixed and constrained mutation (MCM)*, which applies the constraint to multiple image transformation algorithms. The constraint of MCM does not rely on changes in the pixel values. Instead, MCM measures the variation for each transformation algorithm for each mutation result. If each of the variations and the sum of the variations satisfy the constraint, the fuzzer can freely repeat and mix the image transformations. We conducted a user study with 15 participants on each of MNIST and ImageNet datasets to define transformation parameter ranges and enable the measurement of the variations. Although we cannot use whole images of datasets, we generalize the feasibility of transformation parameters through statistical tests. Consequently, the criterion of the oracle was well represented by the transformation parameters. The following summarizes the key contributions of this study:

- **Mixed and constrained mutation.** We propose a new mutation approach for coverage-guided fuzzing in deep-learning systems. We measured variations in the image transformation algorithms and constrained the sum of the variations. As mixed and constrained mutation can combine multiple image transformation algorithms, the fuzzer can mutate inputs freely, but the results still preserve the semantics of their original inputs.
- **Input validity test based on a user study and statistical tests.** Our prime objective is to preserve the semantics of the original inputs. Therefore, we conducted a user study on a certain dataset to define reliable transformation parameter ranges. While the validity of an image was answered by one participant in the previous study, several participants answer the same image in our study. In addition, the feasibility of each transformation parameter is determined by a statistical



(a) Original image (b) Median-Blur:9X9 (c) Scale:1.2, shear:-0.6 (d) Translation:-0.179 (e) Brightness:60, contrast:2.0

Fig. 2. (a) An original image of the MNIST dataset, whose ground-truth label is 9. (b) The result of a 9X9 median blur in the image (a). (c) The result of scaling and shearing image (a) by a factor of 1.2 and 0.6, respectively. (d) The result of translating -5 pixels from image (a). (e) The result of brightening the pixel values of the image (a) by 60 and then contrasting the image by a factor of 2.0. (b) and (c) are invalid images because they violate the ground truth of the original image. (d) and (e) are valid images because still preserve the ground truth of the original image. However, DeepHunter [11] approves (b) but rejects (d). Furthermore, DeepTest [12] generates (c), but it cannot generate (e).

test. Therefore, our mutation does not deviate from the tolerable range of an oracle and can also measure the more precise variation in input. We conducted a second user study and statistical test to evaluate the input validity of the mutation results from the perspective of the oracle.

- **Experiments on various datasets and models.** We verified the performance of the proposed approach using MNIST, STL-10, and ImageNet, which are representative datasets for image recognition. Furthermore, we performed fuzzing on various deep neural network architectures for ImageNet. The fuzzer using the proposed mutation strategy discovered a higher coverage than the existing fuzzing scheme in all experimental settings. In addition, the vulnerability detection performance of the proposed method was found to be several times higher than that of the existing fuzzer.

2. Background

2.1. Coverage-Guided Fuzzing on Deep Learning Systems

Algorithm 1 Coverage-Guided Fuzzing on Deep Learning Models

```
1: **Input:** Corpus: initial seed corpus of test inputs
2: **Output:** V: dataset of vulnerable test cases
3: **Const:** K: batch size
4: while ending condition is not satisfied do
5:    P ← SampleFromCorpus(Corpus)
6:    T ← Mutate(P)
7:    coverage, metadata ← Fetch(T) v 8:    for k ← 1 to K
9:       T_k.add(coverage[k], metadata[k])
10:      if IsMisclassified(metadata[k]) then
11:         V.add(T_k)
12:      else if IsNewCoverage(coverage[k]) then
13:         Corpus.add(T_k)
14:      end if
15:   end for
16: end while
```

Coverage-guided fuzzing is an automated testing method for discovering bugs in a target system (e.g., a deep learning model). The bug of deep learning models is misprediction for input (i.e., the prediction of a model is different from that of a human). The main functionality of fuzzing is the interaction between the model: the fuzzer feeds an input to the model, and the model feeds its internal state about the input back to the fuzzer. As illustrated in Fig. 1, the fuzzer gradually transforms an input that starts from an original input. The internal state of the model is represented through the coverage. Only effective inputs with new coverage are preserved during the fuzzing. Such fuzzing process maximizes the coverage and discovers inputs with unexpected neuron activations.

Basic concept. Algorithm 1 briefly describes the general workflow of coverage-guided fuzzing in deep learning models. The fuzzer starts with an initial seed corpus consisting of clean inputs, and then seeks inputs yielding new coverage by iter-

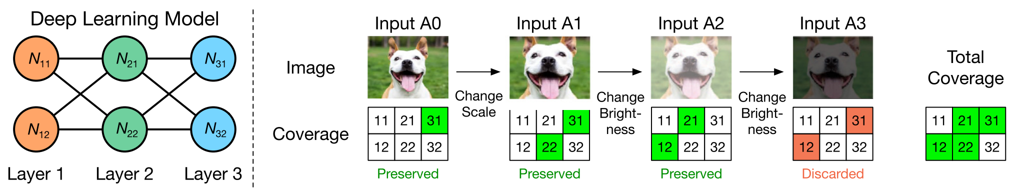


Fig. 1. An example of coverage-guided fuzzing in a deep learning model. Whether each neuron (or section that is a part of activation values of a neuron) is activated is measured through the coverage. The fuzzer aims to construct a test suite that maximizes the number of activated neurons. The fuzzer begins with an original input (A0) and repeatedly generates a new input by mutating a previous input. The new input is preserved if the input activates a neuron that has not been activated by previous inputs. If all of the activated neurons by the new input are already activated by previous inputs, the input is discarded.

atively generating test cases. For each iteration, the fuzzer first samples one seed element as a parent from the seed corpus by using a proper seed selection algorithm (line 5). The fuzzer generates a batch of new test cases by mutating the parent (line 6). The new test cases are fed into the model, and their coverage and metadata are returned (line 7). Coverage indicates the internal behavior of the model, and metadata indicates the prediction result of the model. For each test case of the batch, a new seed element is created with its coverage and metadata (line 9). If a test case triggers a misclassification, the fuzzer adds the test case to a vulnerable dataset to further improve model robustness (lines 10 and 11). Only benign test cases exhibiting new coverage qualify to be added to the seed corpus (lines 12 and 13).

Coverage Criteria. The coverage criterion is a metric used to observe the behavior of a target system about an input. Well-defined coverage criteria for deep learning models have been studied [9,10]. They identify whether each neuron in the model is activated. The higher the coverage of a dataset, the higher the quality that the model. In this study, we consider coverage criteria that are applicable to DeepHunter [11], a representative coverage-guided deep learning fuzzer. Each coverage criterion differs in the definition that the neuron is activated.

- **Neuron Coverage (NC)**, which was proposed in DeepXplore [9], is one of the most fundamental criterion about deep learning coverage. Neuron Coverage defines a certain threshold and identifies the activated neurons that yield activation values over the threshold. The NC measures the ratio of the number of activated neurons to all neurons.
- *Neuron-level coverage criteria* identify whether each neuron has been activated. DeepGauge [10] observed a major-function region or corner-case region of the deep learning model. The major-function region is within the range of the neuron activation values of the training dataset. The corner-case region is outside the major-function region, i.e., it indicates the neuron activation values that are not seen from the training dataset. **K-multisection neuron coverage (KMNC)** separates the major-function region into *k* sections of the same size for each neuron and measures the ratio of the number of sections that have been activated at least once to the number of total sections. **Neuron boundary coverage (NBC)** observes both the upper section and lower section of the corner-case region. **Strong neuron activation coverage (SNAC)** observes only the upper corner section. NBC and SNAC also measure the ratio of the activated sections.
- *Layer-level coverage criteria* identify whether each layer has been activated. **Top-k neuron coverage (TKNC)** selects top-k neurons with the highest activation values in each layer for an input. TKNC measures the ratio of neurons that have been selected at least once to all neurons.

2.2. Input mutation

Mutation is a module that directly affects the fuzzer and triggers the vulnerability of the target system. The fuzzer performs an appropriate mutation function for a given input to change the behavior of the system while preserving the semantics of the input.

Image transformation algorithms. Image transformation algorithms are used in input mutation to implement real-world changes such as different filming or weather conditions. For instance, as the change from the leftmost image of Fig. 2 to right images of Fig. 2, various new images can be generated from the image by image transformation. DeepTest [12] considered three types of image transformation algorithms: affine, linear, and convolution transformations. Affine transformations, such as translation, shearing, scaling, and rotation, change the location of pixels while preserving their values. This change is difficult to implement using the gradient of the model; however, we can utilize image transformation library. Linear transformations, such as adjusting brightness and contrast, change pixel values with a constant rule. It is well known that even a small linear change in the input layer can lead to significant changes in the model output [15]. Convolution transformations, such as blur, perform a convolution operation. This operation offers an effect similar to that of deploying a convolutional layer before the input of the model.

Previous coverage-guided fuzzers, DeepTest [12] and DeepHunter [11], designed their mutation strategies upon the three types of image transformation algorithms. In the following, we summarize those fuzzing schemes.

DeepTest. DeepTest demonstrated that different image transformation algorithms activate different coverage. It also showed that the compound of image transformation algorithms further increases the coverage. Since the search space of all possible combinations is tremendous, DeepTest performs neuron-coverage-guided greedy search. For each fuzzing iteration, one previous input is popped from the seed corpus and then two selected transformation algorithms are used simultaneously to generate a new input: one is selected from the prioritized transformation set and the other is randomly selected. If the new input increases coverage, the input is added to the seed corpus, and the used transformations are added to the prioritized set. DeepTest adopts a depth-first approach, that is, the popped input is not added again to the seed corpus.

DeepHunter. DeepHunter consists of three main components: prioritized seed selection, metamorphic mutation, and coverage criterion. DeepHunter repeats those components, following Algorithm 1. For the seed selection, DeepHunter assigns a probability of being selected to each seed input. The more frequently an input has been selected, the lower probability the input has. The metamorphic mutation of DeepHunter combines image transformation algorithms of DeepTest and randomized pixel value transformation. DeepHunter focuses on valid test cases that preserve the semantics of their original inputs. For this purpose, it prohibits the combination of image transformation algorithms. The pixel value transformation can be repeated unlimitedly, but the result is constrained within l_p-ball. One of the multiple coverage criteria in Section 2.1 can be utilized to measure the freshness of the mutated input, which enables testing the model at a multi-granularity level.

Problem of DeepTest and DeepHunter. Although DeepTest and DeepHunter are effective approaches for generating mutant images with new coverage, they hinder the reliability of fuzzing because they do not consider real-world constraints. An unstable constraint causes the fuzzer to generate invalid inputs that deviate from the real-world distribution. The image in Fig. 2(b) is valid for the constraint of DeepHunter, but it is unrecognizable to human eyes because the shape of the digit has been crushed. If a constraint is defined very loosely, invalid inputs are easily generated when the multiple transformation algorithms overlap. In Fig. 2(c), each transformation algorithm uses valid parameters for DeepHunter, but it is ambiguous to preserve the class of its original image in the resulting image because the image is confused with eight. Although multiple transformation algorithms must be overlapped to create diverse inputs, the input variation should be limited by a precise constraint to hold the input on a real-world distribution.

Some transformation parameters of DeepHunter limit the variation more than humans can allow, thus the number of possible inputs is also limited. In Fig. 2(d), the variation in the image deviates from the constraint of DeepHunter, but the class of its original image is still preserved because the shape of the digit has not been damaged. Even though the transformations overlap, the mild overlap based on valid parameters has a higher possibility of generating valid inputs. Fig. 2(e) still shows the visual ground truth of the original image. In addition, the transformations applied to Fig. 2(e) used parameters that deviate from the constraint of DeepHunter, but these parameters are valid from the perspective of humans. Therefore, overly tight mutation should be relieved to increase the diversity of the resulting images.

3. Methodology

3.1. Mixed and constrained mutation

In this section, we describe our mutation strategy, called *mixed and constrained mutation (MCM)*. MCM is applied to the coverage-guided deep learning fuzzing as a mutation function of line 6 in Algorithm 1. Because we focus on a new mutation methodology in this work, we use DeepHunter as a base fuzzing framework. That is, we use other fuzzing modules, such as seed selection and coverage measurement, the same as DeepHunter.

Algorithm 2 Mixed and Constrained Mutation

```
1: **Input:** $P$: parent test case
2: **Output:** $T$: mutated test cases
3: **Const:** $K$: batch size
4: **Const:** $N$: maximum try number for a successful mutation
5: **Const:** $C$: constraint for the sum of mutation magnitude
6: **Const:** $S$: set of transformation algorithms
7: **for** $k \leftarrow 1$ to $K$ **do**
8:     $x_0^k \leftarrow P$.data
9:     $v^k \leftarrow P$.variations
10:    **for** $n \leftarrow 1$ to $N$ **do**
11:        $tid \leftarrow$ randomSelectTransform($S$)
12:        $p \leftarrow$ randomSelectParam($S[tid]$)
13:        $x^k \leftarrow S[tid](x_0^k, p)$
14:        $v^k[S[tid]] \leftarrow$ update($v^k[S[tid]], p$)
15:        $\eta(v^k) = \{\eta(v^k[\tau]) | \tau \in S\}$
16:        **if** $all(\eta(v^k) \leqslant 1)$ and $sum(\eta(v^k_{affine})) \leqslant C$ and $sum(\eta(v^k_{linear})) \leqslant C$ **then**
17:            $e \leftarrow \{x^k, v^k\}$
```

(*continued on next page*)

a (*continued*)

```
Algorithm 2 Mixed and Constrained Mutation
18:         $T$.add($e$)
19:         break
20:       end if
21:    end for
22: end for
23: return $T$
```

3.1.1. Mutation overview

Algorithm 2 represents the operation of the mixed and constrained mutation (MCM). The mutation function obtains a parent seed element P for seed selection and returns a batch of mutated test cases T. The parent contains the information about its data x_0 (i.e., current image) and variations v (i.e., accumulated transformation parameters from its original image to the current shape). The process of generating one mutant is between line 8 and line 21 of Algorithm 1. If one of the transformation algorithms except for random noise has already been selected for the parent in DeepHunter, those transformation algorithms cannot be selected afterward. However, in line 11 of Algorithm 2, our mutation function can freely select transformation algorithms. One transformation algorithm is accessed by $S[tid]$ where S is a set of transformation algorithms and tid is an index for the selected transformation algorithm. In line 12, the function selects the transformation parameter p for the selected algorithm. The range of possible parameters is defined by a user study and statistical tests (see §3.2.2). Line 13 performs the transformation and generates a mutated input x^k. The variation v^k in the mutant for the selected transformation is updated in line 14 and then normalized by the normalization function η in line 15. Line 16 verifies whether the variations of the mutant satisfy the constraint of mutation C. Our approach is different from DeepHunter in that DeepHunter uses the l_0 and l_p-norm as the only constraints for random noise, while we apply another constraint for three types of transformations (e.g., affine, linear, and convolution). The successful mutation means a case where the mutation returns a mutated input that satisfies the constraint within the maximum number of attempts N. If the mutation was successful, the input is returned immediately. Otherwise, the function attempts the mutation again within N. The proposed mutation method can generate several mutants from one seed element. Given a batch size K, the function performs from line 8 to line 21 for K times and returns K mutants.

3.1.2. How to constrain variations

In the variation update stage of line 14, we multiply the variation of transformation algorithms, such as scale and contrast, whose baseline of parameter range is 1, by the applied transformation parameter. We add a variation of the other transformation algorithms, whose baseline of parameter range is 0 by the applied transformation parameter. Updating the variation facilitates the repeated transformation of the same algorithm. As repeated blurring seriously distorts the image, we maximize the variation with one-blur execution. The transformation algorithms with baseline 0 can yield a negative variation. Instead, in line 15, the variations are normalized to $0 \sim 1$ using the normalization function η in Eq. 1. Here, $b[\tau]$ is the baseline parameter the transformation τ, and $u[\tau]$ and $l[\tau]$ are the upper and lower bounds of the parameter range of τ, respectively. In line 16, the function $all(\eta(v))$ verifies whether each variation does not exceed the maximum normalization value of 1. This verification method prevents one transformation from repeating excessively. At the same time, line 16 also verifies that the summation of variations $sum(\eta(v))$ satisfies constraint value C. The summation of variations is measured as a precaution against the phenomenon in which a mutated input seems invalid, even though all of its normalized variations of transformations are valid. In particular, the overlap of the same type of transformation intensifies distortion. Therefore, we apply a constraint value for the summation of variations for each affine transformation and linear transformation. In this study, we set the constraint value as $C = 1.5$.

$$\eta(v[\tau]) = \begin{cases} (v[\tau] - b[\tau])/(u[\tau] - b[\tau]), & \text{if } v[\tau] \geqslant b[\tau] \\ (b[\tau] - v[\tau])/(b[\tau] - l[\tau]), & \text{otherwise} \end{cases} \tag{1}$$

3.2. Transformation Parameters based on User Study

3.2.1. Study design

Because our mutation approach combines multiple transformation algorithms, each transformation parameter should be feasible in the real-world. We conducted a user study and statistical test to define the transformation parameters that reflect the real-world changes. Our study targeted the MNIST, STL-10, and ImageNet datasets, considering both grayscale and RGB channel images. The input dimensionalities of MNIST, STL-10, and ImageNet are 28×28×1, 96×96×3, and 224×224×3, respectively. We excluded CIFAR-10 dataset in our study because it consists of low resolution images whose classes may be confusing to humans.

We verified whether the semantics of the images before and after transformation were identical for a transformation parameter through a t-test. A survey was used to collect samples for statistical testing. We collected answers regarding both the original images and their transformation results. It is practically impossible to survey whole images in a dataset. Nevertheless, our results on a part of images can be generalized to other images with the aid of statistical tests.

For each dataset, the participants completed the survey five times over five days. They answered the original images on the first day and then answered the transformed images on the following days. The user study was a lab study of 15 participants, who were separated into three groups of five people. We defined the questionnaire parameter ranges based on our manual search and DeepHunter parameter ranges. The questionnaire parameters are described in the column named "*Questionnaire*" of Table 1–3. The number of transformation parameters used for the survey was 51, 51 and 60 for MNIST, STL-10 and ImageNet, respectively.

Although both our study and DeepHunter perform a user test, they are different for three reasons. First, five participants answered one image in our study, while each image was answered by one participant in DeepHunter, where results can be biased. Second, the validity of transformed images was compared with their original images in our study, but DeepHunter did not consider original images. Finally, we measured the accuracy of each image and then determined the feasibility of each transformation parameter through a statistical test, but DeepHunter measured the ratio of correctly recognized inputs to all mutated inputs as the validity of their mutation strategy.

On the MNIST and STL-10 datasets, participants answered 20 transformed images for each parameter. Thus, we collected 60 samples for each parameter. On the ImageNet dataset, the participants answered 12 images, and we collected 36 samples for each parameter. The total number of images that the participants answered was 1,120 on MNIST and STL-10 (i.e., 100 original images and 1,020 transformed images) and 900 on ImageNet (i.e., 180 original images and 720 transformed images). As we composed three groups, the total number of images used in the user study was 3,360 on MNIST and STL-10 and 2,700 on ImageNet, respectively. We used all the classes on MNIST and STL-10, but we used 10 representative classes among 1,000 classes for each group on ImageNet.

3.2.2. Study results

We conducted a statistical test using the samples collected through the survey. Therefore, we performed a t-test for each transformation parameter. One sample of the t-test consisted of a pair of accuracies for the original image and its transformed image. The significance level, α, was set to 5%. The null hypothesis was that the accuracy of the original images is higher than that of the transformed images for a transformation parameter. We consider a transformation parameter to be valid only if its p-value is greater than 0.05. To define a rigorous parameter range, we consider that if a mild parameter value of one transformation algorithm is statistically invalid, a more intense parameter is also invalid even though it is statistically valid. For instance, if scaling an image by 1.2 times larger is statistically invalid, scaling the image by 1.5 times larger is also invalid, even though its statistical result is valid.

The range of transformation parameters defined by DeepHunter and our statistical test results are presented in the column "*DeepHunter*" and the column "*Test Result*" in Table 1–3, respectively. In Table 1 and 2, the parameter ranges of all the transformation algorithms of the test result are different from those of DeepHunter. For the results on ImageNet in Table 3, the parameter ranges of all transformation algorithms changed except for shear.

We can be more generous with brightness and contrast on all datasets than DeepHunter. For instance, darker images in MNIST dataset can be recognized as valid since all images in this dataset show only a black background and a white digit. However, the semantics of an MNIST image are largely harmed when a part of the digit is occluded by scale, shear, and rotation. Therefore, the parameter ranges of such transformations on MNIST are reduced. On the other hand, most of the parameter ranges of STL-10 and ImageNet datasets become looser than those of DeepHunter since participants can recognize images even if some part of images are occluded. DeepHunter fixed the number of pixels that can be moved by translation regardless of the image size so that mutation is not effective on datasets of larger images such as STL-10 and ImageNet. Therefore, we changed the parameter ranges of translation from the number of movable pixels to the ratio to the image size, and we extended the parameter ranges for STL-10 and ImageNet.

Because the kernel size of the blur in DeepHunter is relatively large for the image size of MNIST, as shown in Fig. 2(b), the number of possible parameters for blur is also reduced. On the other hand, larger parameters for blur are allowed in STL-10 and ImageNet datasets. Some transformations yielded the same parameter range as that of the questionnaire. The resulting parameter ranges could be extended by extending the questionnaire parameter ranges; however, we maintained the current result for the rigorous constraint.

4. Evaluation

To evaluate the effectiveness of the proposed mutation strategy, which we called the mixed and constrained mutation (MCM), we established three research questions.

RQ1. For efficiency of fuzzing, does the MCM generate the larger number of statistically valid test inputs compared to the previous mutation strategy with the same fuzzing iterations?.

RQ2. Does coverage-guided fuzzing based on MCM discover more diverse seed elements and coverage on various datasets and deep learning architectures than the existing fuzzing scheme?.

Table 1
Parameter ranges for transformation algorithms (MNIST dataset).

Transformation	Type	DeepHunter	Questionnaire	Test Result
Translation	Affine	$-0.107 \leqslant p \leqslant 0.107$	$-0.179 \leqslant p \leqslant 0.179$	$-0.179 \leqslant p \leqslant 0.107$
Scale		$0.7 \leqslant p \leqslant 1.2$	$0.5 \leqslant p \leqslant 1.5$	$0.5 \leqslant p \leqslant 1.1$
Shear		$-0.6 \leqslant p \leqslant 0.6$	$-0.6 \leqslant p \leqslant 0.6$	$-0.25 \leqslant p \leqslant 0.25$
Rotation		$-50^\circ \leqslant p \leqslant 50^\circ$	$-50^\circ \leqslant p \leqslant 25^\circ$	$-25^\circ \leqslant p \leqslant 10^\circ$
Contrast	Linear	$0.5 \leqslant p \leqslant 1.3$	$0.1 \leqslant p \leqslant 10.0$	$0.3 \leqslant p \leqslant 2.5$
Brightness		$-20 \leqslant p \leqslant 20$	$-100 \leqslant p \leqslant 100$	$-100 \leqslant p \leqslant 60$
Blur	Comvolution	$3 \leqslant p \leqslant 5$	$3 \leqslant p \leqslant 4$	$p = 3$
GaussianBlur		$3 \leqslant p \leqslant 7$	$3 \leqslant p \leqslant 5$	$p = 3$
MedianBlur		$3 \leqslant p \leqslant 9$	$3 \leqslant p \leqslant 9$	$p = 3$

Table 2
Parameter ranges for transformation algorithms (STL-10 dataset).

Transformation	Type	DeepHunter	Questionnaire	Test Result
Translation	Affine	$-0.031 \leqslant p \leqslant 0.031$	$-0.208 \leqslant p \leqslant 0.208$	$-0.104 \leqslant p \leqslant 0.208$
Scale		$0.7 \leqslant p \leqslant 1.2$	$0.5 \leqslant p \leqslant 1.7$	$0.5 \leqslant p \leqslant 1.5$
Shear		$-0.6 \leqslant p \leqslant 0.6$	$-0.6 \leqslant p \leqslant 0.6$	$-0.6 \leqslant p \leqslant 0.5$
Rotation		$-50^\circ \leqslant p \leqslant 50^\circ$	$-60^\circ \leqslant p \leqslant 60^\circ$	$-60^\circ \leqslant p \leqslant 60^\circ$
Contrast	Linear	$0.5 \leqslant p \leqslant 1.3$	$0.1 \leqslant p \leqslant 2.0$	$0.2 \leqslant p \leqslant 2.0$
Brightness		$-20 \leqslant p \leqslant 20$	$-90 \leqslant p \leqslant 90$	$-70 \leqslant p \leqslant 90$
Blur	Comvolution	$3 \leqslant p \leqslant 5$	$2 \leqslant p \leqslant 9$	$2 \leqslant p \leqslant 5$
GaussianBlur		$3 \leqslant p \leqslant 7$	$3 \leqslant p \leqslant 11$	$3 \leqslant p \leqslant 9$
MedianBlur		$3 \leqslant p \leqslant 9$	$3 \leqslant p \leqslant 5$	$3 \leqslant p \leqslant 5$

Table 3
Parameter ranges for transformation algorithms (ImageNet dataset).

Transformation	Type	DeepHunter	Questionnaire	Test Result
Translation	Affine	$-0.013 \leqslant p \leqslant 0.013$	$-0.156 \leqslant p \leqslant 0.156$	$-0.156 \leqslant p \leqslant 0.156$
Scale		$0.7 \leqslant p \leqslant 1.2$	$0.4 \leqslant p \leqslant 1.7$	$0.4 \leqslant p \leqslant 1.7$
Shear		$-0.6 \leqslant p \leqslant 0.6$	$-0.6 \leqslant p \leqslant 0.6$	$-0.6 \leqslant p \leqslant 0.6$
Rotation		$-50^\circ \leqslant p \leqslant 50^\circ$	$-80^\circ \leqslant p \leqslant 80^\circ$	$-80^\circ \leqslant p \leqslant 80^\circ$
Contrast	Linear	$0.5 \leqslant p \leqslant 1.3$	$0.2 \leqslant p \leqslant 1.8$	$0.4 \leqslant p \leqslant 1.8$
Brightness		$-20 \leqslant p \leqslant 20$	$-80 \leqslant p \leqslant 80$	$-80 \leqslant p \leqslant 80$
Blur	Comvolution	$3 \leqslant p \leqslant 5$	$2 \leqslant p \leqslant 9$	$2 \leqslant p \leqslant 7$
GaussianBlur		$3 \leqslant p \leqslant 7$	$3 \leqslant p \leqslant 11$	$3 \leqslant p \leqslant 11$
MedianBlur		$3 \leqslant p \leqslant 9$	$3 \leqslant p \leqslant 5$	$3 \leqslant p \leqslant 5$

RQ3. Does coverage-guided fuzzing based on MCM yield the better performance of vulnerability detection than the existing fuzzing scheme in terms of the quantity and diversity?.

4.1. Experimental setup

We performed an evaluation on MNIST, STL-10, and ImageNet, which are representative image-recognition datasets. LeNet5 [16], the classifier for the MNIST dataset, was trained using 60,000 images in the training dataset. We trained ResNet50 [17] using 5,000 images to classify the STL-10 dataset with the classification power. For the ImageNet classifier, we targeted four types of deep learning architectures. We used a pre-trained model of MobileNet [18], Inception-v3 [19], ResNet50, and ViT-L/32 [20] from Keras. For MNIST and STL-10, we used 1,000 original images, which were correctly classified by the model, in the test dataset as the initial seeds of fuzzing. For ImageNet, we used 100 original and benign images in the validation dataset as initial seeds. Classes of the initial seeds were distributed uniformly on MNIST and STL-10, and each of the initial seeds on ImageNet had a different class.

We mainly compared the fuzzing performances of MCM and DeepHunter [11]. We excluded DeepTest, which allowed the combination of multiple transformation algorithms but did not apply the constraint for those algorithms. We applied NC, KMNC, and NBC to LeNet5 of MNIST, ResNet50 of STL-10, and MobileNet of ImageNet as the coverage criteria of the fuzzer. We set their coverage parameters to 0.75, 1,000, and 10 respectively for LeNet5 and MobileNet. This setting was the same as that used in the original work of DeepHunter. Because the parameter size of ResNet50 is considerably large, we set the coverage parameter of KMNC for STL-10 as 10. To measure KMNC and NBC, we needed to measure neuron activations of the training dataset beforehand, but it was difficult to perform the measurement because of the scale of the ImageNet training dataset[1]. Therefore, we applied only NC to Inception-v3, ResNet50, and ViT-L/32 of ImageNet, with the same parameters as DeepHunter. We used the constraint value $C = 1.5$ for image transformations, including the affine and linear algorithms, as described in Section 3.1. For pixel value transformations we used l_0-norm and l_∞-norm, and we set the constraint value as 0.05 and 0.3, respectively. We set the batch size to $K = 4$ and the maximum number of attempts to $N = 50$ for the mutation function. If the mutation returns a valid input within the maximum number of attempts, we refer the mutation succeeds. We used the probabilistic seed-selection algorithm designed by DeepHunter. We reported the result of four hours of fuzzing execution for all datasets and model architectures. Further, the fuzzers were repeated five times, and we report the average values for all experiments.

4.2. Results

4.2.1. RQ1. Input Validity

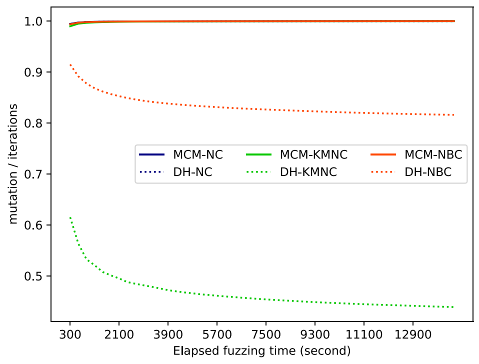
Fig. 3 illustrates the ratio of successful mutations to iteration during fuzzing execution. Only the mutated input from a successful mutation can be fed into the model. Therefore, as the mutation success rate increases, the fuzzer feeds more inputs to the model. As a consequence, the fuzzer gets more opportunities to search the broader coverage and more adversarial examples. MCM, our proposed approach, resulted in the higher mutation success rate than DeepHunter for all models and coverage criteria. DeepHunter exceptionally achieved almost a 100% mutation success rate for NC of LeNet5. This is because NC on LeNet5 was stuck in the initial seed corpus, as shown in Table 4. In other words, transformation algorithms never overlap, thus mutation results are always valid. The success rate of DeepHunter is reduced as iteration increases when new seed elements have been discovered in some degree. Particularly, DeepHunter had more difficulty for KMNC on LeNet5 and MobileNet, adding many more seed elements. For the same reason, DeepHunter had a lower mutation success rate with NBC on ResNet50 of STL-10. DeepHunter started fuzzing with 100% of mutation success rate for all coverage criteria on STL-10, but its success rate reduced consistently. For that case, DeepHunter yielded 92.27%, 91.13%, and 71.8% of mutation success rate with NC, KMNC, and NBC at the end of fuzzing. Moreover, the mutation success rate of DeepHunter on other models was lower than that on STL-10. DeepHunter finished fuzzing with 43.9% and 81.6% of success rate with KMNC and NBC on LeNet5, respectively. It also yielded from 60% to 70% of mutation success rate on models of ImageNet. Surprisingly, MCM maintained 100% of mutation success rate consistently within 50 of the maximum try number on LeNet5 of MNIST and ResNet50 of STL-10. The mutation success rate of MCM reached almost 100% on models of ImageNet as fuzzing proceeds.

DeepHunter has a low mutation success rate because it easily concludes that a mutant is invalid due to its limited transformation choices and constraint. Once one of affine, linear, and convolution transformation algorithms is applied to an input, DeepHunter can only use random noise afterward. In this situation, the l_p-norm for the input increases consistently and is saturated at sometime. A mutant generated from the saturated input is always invalid and rejected by DeepHunter. On the other hand, the MCM is free to use multiple transformation algorithms. Consequently, it can select another algorithm even if the variation in one transformation algorithm is maximized. MCM can further improve its performance by selecting transformation algorithms adaptively according to their variations, but we leave this issue for future work.

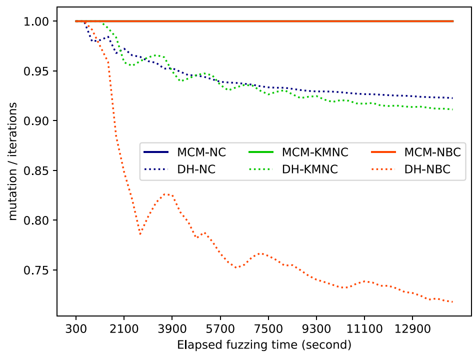
Fig. 4 presents the distribution of variations of seed elements and adversarial examples as a mutation result of KMNC on LeNet5, ResNet50 of STL-10, and MobileNet. Mutants of DeepHunter generate a highly limited variation distribution on the left of the vertical line, which indicates its constraint, in Fig. 4. On the other hand, as shown in the figure, mutants of MCM generated a broad variation distribution within our constraint $0 \leqslant C \leqslant 4$. The variation value of 1.0 can be produced by using one transformation algorithm with a maximized parameter. As a result, the probability on this value is higher than other variations. While both seed elements and adversarial examples are distributed throughout the variation range, the probability of seed elements was higher than that of adversarial examples on smaller variation values. In contrast, the probability of adversarial examples was higher than that of seed elements on larger variation values. It is likely to be due to that adversarial examples were transformed once more from their parents, i.e. seed elements. Mutants of MNIST tend to have larger variations than those of other datasets. As the model for MNIST is lightweight, MCM processed more iterations and its mutants had higher generations. We found that seed elements and adversarial examples on STL-10 rarely exist in the variation value from 3.0 to 4.0. It is because the parameter size of ResNet50, the target model of STL-10, is large. Although the fuzzing on such large model is relatively slow, the probability on larger variation values would increase if the fuzzing goes further.

User study and statistical test on mutation results. We verified whether the mutated inputs distributed in the variation section of Fig. 4 are valid by the oracle. We performed the final user study and statistical tests with the same inputs, as shown in Fig. 4. The survey was carried out in the same manner as described in Section 3.2. We separated the 15 participants into three groups. Participants answered 100 original images on the first day, and answered the mutated images selected from each variation section after that day. We asked about different original images of MNIST and STL-10 for each group, but we asked the same original images of ImageNet for all groups. The size of each variation section was 0.1 for the MNIST and 0.2 for STL-10 and ImageNet. We selected 60 seed elements or adversarial examples as questionnaire images of one variation section, and separated them into three groups. We verified the entire variation ranges of MNIST and ImageNet. However, we verified mutated images of STL-10 whose variations are less than 3.0, since we could not collect enough images with

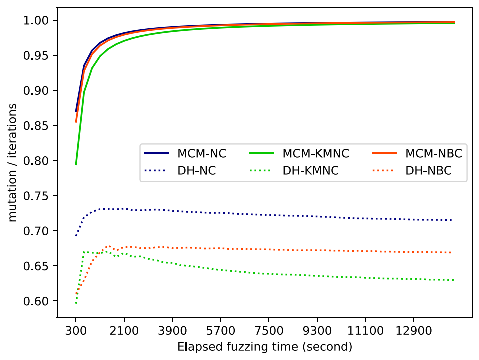
[1] Neuron activation of the training dataset on MobileNet was provided by DeepHunter [11]



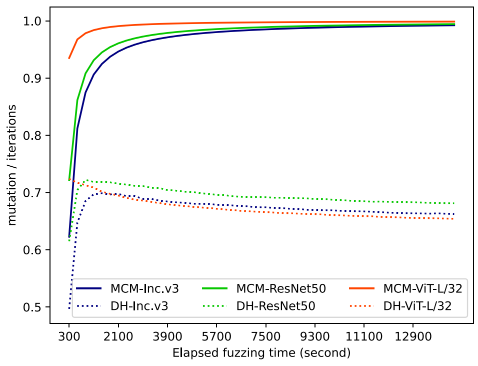
(a) LeNet5 (MNIST)



(b) ResNet50 (STL-10)



(c) MobileNet (ImageNet)



(d) Inception-v3, ResNet50, and ViT-L/32 (ImageNet, NC only)

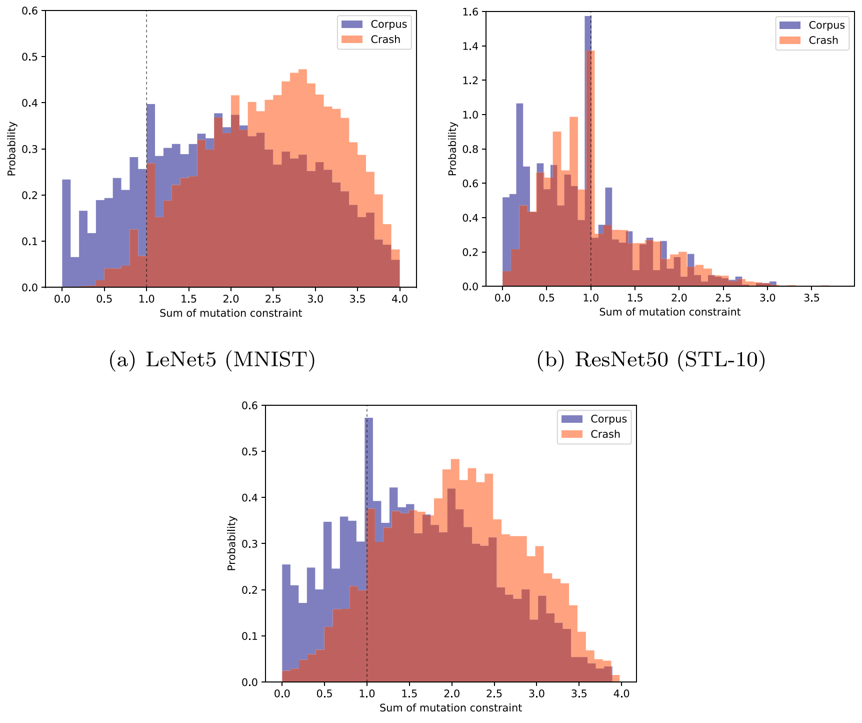
Fig. 3. The ratio of successful mutations to fuzzing iteration. For each time the fuzzer executes from line 5 to line 15 of Algorithm 1, the number of iterations increases by one. For each time the fuzzer generates a valid mutation result with the constraint at one iteration, the number of successful mutations increases by one. DH in the legend denotes DeepHunter. The number of iterations and successful mutations was accumulated at the start of fuzzing. LeNet5 was tested on the MNIST dataset, and ResNet50 was tested on the STL-10 and ImageNet datasets. The other models were tested on the ImageNet dataset. LeNet5, ResNet50 for STL-10, and MobileNet were tested using NC, KMNC, and NBC. Inception-v3, ResNet50 for ImageNet, and ViT-L/32 were tested using only the neuron coverage.

Table 4
The number of seed elements with new coverage. The results were averaged over five fuzzing executions, and the number of initial seeds was subtracted. The column MCM is the approach proposed in this study. DH denotes DeepHunter. LeNet5 was tested on the MNIST dataset, and ResNet50 was tested on the STL-10 and ImageNet datasets. The other models were tested on the ImageNet dataset. LeNet5, ResNet50 for STL-10, and MobileNet were tested using NC, KMNC, and NBC. Inception-v3, ResNet50 for ImageNet, and ViT-L/32 were tested using only the neuron coverage.

Coverage	**LeNet5**		**ResNet50 (STL-10)**		**MobileNet**		**Inception-v3**		**ResNet50 (ImageNet)**		**ViT-L/32**	
Criteria	**MCM**	**DH**	**MCM**	**DH**	**MCM**	**DH**	**MCM**	**DH**	**MCM**	**DH**	**MCM**	**DH**
NC	31.2	31.0	1,329.0	1,052.2	916.0	766.4	1,536.0	1,129.4	1,354.0	965.2	569.8	586.4
KMNC	77,648.4	69,395.8	1,421.2	1,298.8	4,745.8	3,361.4	N/A	N/A	N/A	N/A	N/A	N/A
NBC	903.6	851.2	10,455.8	9,131.0	6,421.4	5,940.6	N/A	N/A	N/A	N/A	N/A	N/A

variations over 3.0 for the statistical test. The total number of images that a participant answered was 900 on MNIST, 400 on STL-10, and 500 on ImageNet, respectively. Further, the total number of images used for the survey was 2,700 on MNIST, 1,200 on STL-10, and 1,300 on ImageNet.

Fig. 5 summarizes the result of the statistical test on the result of survey. We carried out a separated t-test for each variation section. The null hypothesis was that the accuracy of original images is higher than that of mutated images for a variation section. For each image, we derived the accuracy by calculating the ratio of participants who correctly answered the



(a) LeNet5 (MNIST)

(b) ResNet50 (STL-10)

(c) MobileNet (ImageNet)

Fig. 4. The probability distribution of the total variation of seed elements (corpus) and adversarial examples (crash) discovered by mixed and constrained mutation. The vertical dashed line on $x = 1$ represents the constraint of DeepHunter. We report the result of KMNC as a representative coverage criterion.

image to total participants. After that, the accuracy of one variation section was measured by averaging the accuracy of images in the section. In Fig. 5, the accuracy that the answered class and the original class coincide was above 94% for all variation sections. This result appeared for both original images and mutated images on all datasets. If the accuracy of mutated images are higher than that of their original images, the p-value is above 0.5. Otherwise, the p-value is below 0.5. As mutated images become more accurate than their original images, the p-value converges into 1. Contrarily, as original images become more accurate than mutated images, p-value converges into 0. It is interesting that variation sections frequently exist where the accuracy of mutated images was higher than that of original images. One of the possible reasons is that participants answered the mutated images after the original images, and they learned the survey images after answering mutated images. However, most of all, this phenomenon indicates that the semantics of mutated images were not damaged. Although there is a deviation of the p-value among variation sections, the p-value maintained its high value for large variations on MNIST. On STL-10 and ImageNet, the p-value decreased in general as the variation increases. Especially, STL-10 showed p-values of almost 1 for small variations but the amount of reduction was steeper than ImageNet. It is because the accuracy of original images of large variations was near 100% even though the accuracy of mutated images was also high. Images with large variations imply that they underwent more generations during fuzzing. In other words, original images of large variations are visually more stable so that the model is more robust to those images. It should be noted that the accuracy of a variation section considerably depends on the appearance of included images. There were some images that multiple participants answered incorrectly. According to whether the ambiguous images were included in the variation section, the p-value oscillated throughout variations. Nevertheless, the p-value was always above the significance level $\alpha = 0.05$ on all datasets.

We also performed additional statistical tests to identify that the degree of distortion is similar between seed elements and adversarial examples. The used samples were the same as in the t-test. We compared the accuracies of original images and mutated images. We performed Mann Whitney U-test, a non-parametric test because the number of used samples was less than that for the t-test. The results are described in Fig. 6. For all datasets, there was no consistent difference in p-values between seed elements and adversarial examples: p-values irregularly oscillated throughout variations. This phenomenon demonstrates that the recognizability of our mutation results depends not on the type of a mutant or its variation scale, but on the shape of its original image. Above all things, p-values were always larger than the significance level, except for seed elements in MNIST for variation 3.9 with a minutely lower p-value. In conclusion, the mixed and constrained mutation statistically preserves the semantic of original inputs.

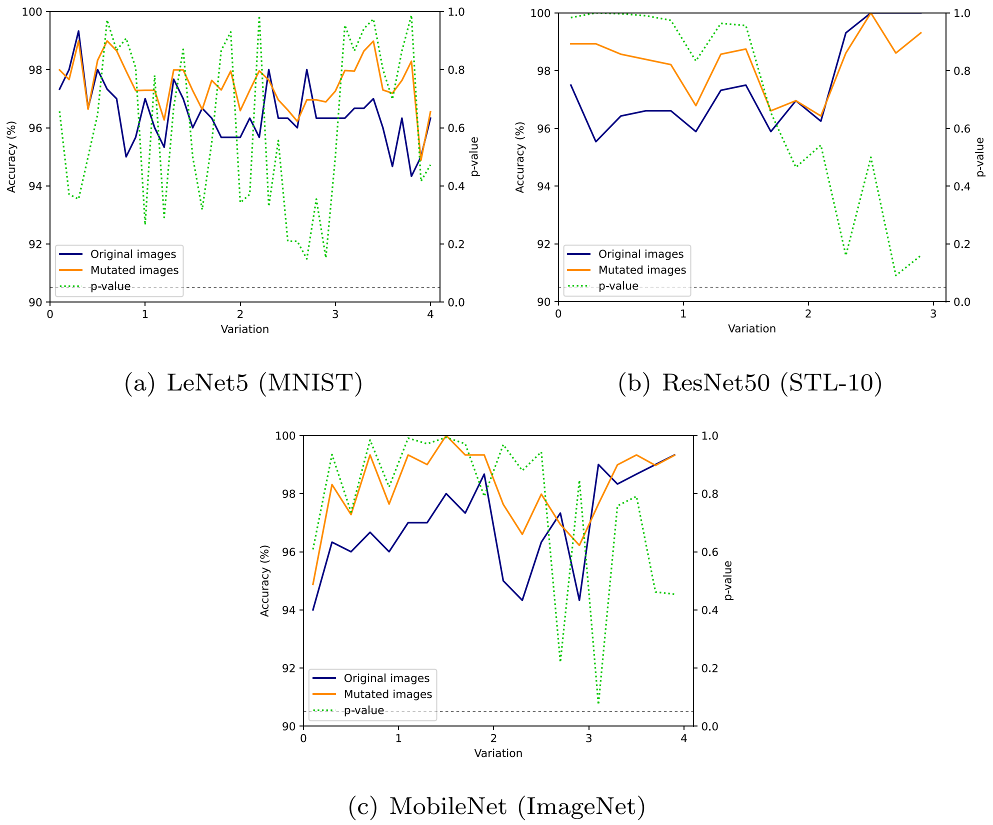


Fig. 5. The result of the final user study and t-test on mutated images generated by mixed and constrained mutation. Seed elements and adversarial examples were simultaneously used to verify the same variation section. The accuracies of the original and mutated images are plotted by solid lines, and the p-value is plotted by a green dashed line. The horizontal dashed line indicates the significance level, $\alpha = 0.05$. The accuracy of the original images can be unstable for varying sections because a set of images is different for different sections.

4.2.2. RQ2. Seed Elements with New Coverage

We identified that the performance of the deep learning model verification was also enhanced due to the improved in the mutation module. Table 4 summarizes the number of total seed elements after the end of fuzzing execution. On LeNet5, ResNet50 of STL-10, and MobileNet, MCM discovered more seed elements than DeepHunter for all coverage criteria. On LeNet5 and MobileNet, the difference between two fuzzers appeared larger for KMNC than NBC and NC. This is because NBC has difficulty activating corner-case neuron values and NC becomes saturated with few inputs for those models. This phenomenon gets worse on a small scale model such as LeNet5. However, on ResNet50 of STL-10, the difference between two fuzzers appeared larger for NBC and NC than KMNC. We see that STL-10 has few train images than other datasets so that their neuron activation values cannot construct a wide major function region. For the quantitative analysis, DeepHunter detected 69,395.8 seed elements using KMNC, while MCM detected 12% more seed elements than DeepHunter. DeepHunter discovered 1,052 and 9,131 seed elements, but MCM discovered 1,329 and 10,455 seed elements, which correspond to 26% and 14% larger amounts, respectively. DeepHunter discovered 766.4, 3,361.4, and 5,940.6 seed elements on MobileNet using NC, KMNC, and NBC. On the other hand, MCM discovered 20%, 41%, and 8% more seed elements than DeepHunter, respectively. On two other ImageNet models, Inception-v3 and ResNet50, DeepHunter discovered 1,129.4 and 965.2 seed elements, but MCM discovered 36% and 40% more seed elements. In summary, the average number of seed elements with new coverage was 17.6% higher than DeepHunter across all environments.

The MCM discovered slightly fewer seed elements than DeepHunter on ViT-L/32. However, we do not determine the diversity of seed elements using only their populations. Therefore, we compare the coverage rate of the fuzzers in Fig. 7. In Fig. 7(d), the MCM achieved higher coverage rate than DeepHunter on ViT-L/32. This result indicates that although DeepHunter discovered more seed elements, the seeds yielded similar neuron activations. For all coverage criteria and models, except the NC of LeNet5, the MCM maintained higher coverage throughout the fuzzing execution. For instance, DeepHunter achieved 89.12%, and 17.42% of coverage rate with KMNC and NBC on LeNet5, but MCM achieved 3.82%p, and 1.24%p higher coverage rate than DeepHunter. We also see that the coverage rate of NC was saturated at an early stage of fuzzing on LeNet5. The coverage rate of NC was largely reduced on ResNet50 of STL-10. On this dataset, DeepHunter achieved 12.99%, 99.48%, and 32.92% of coverage rates with NC, KMNC, and NBC. MCM achieved slightly higher coverage

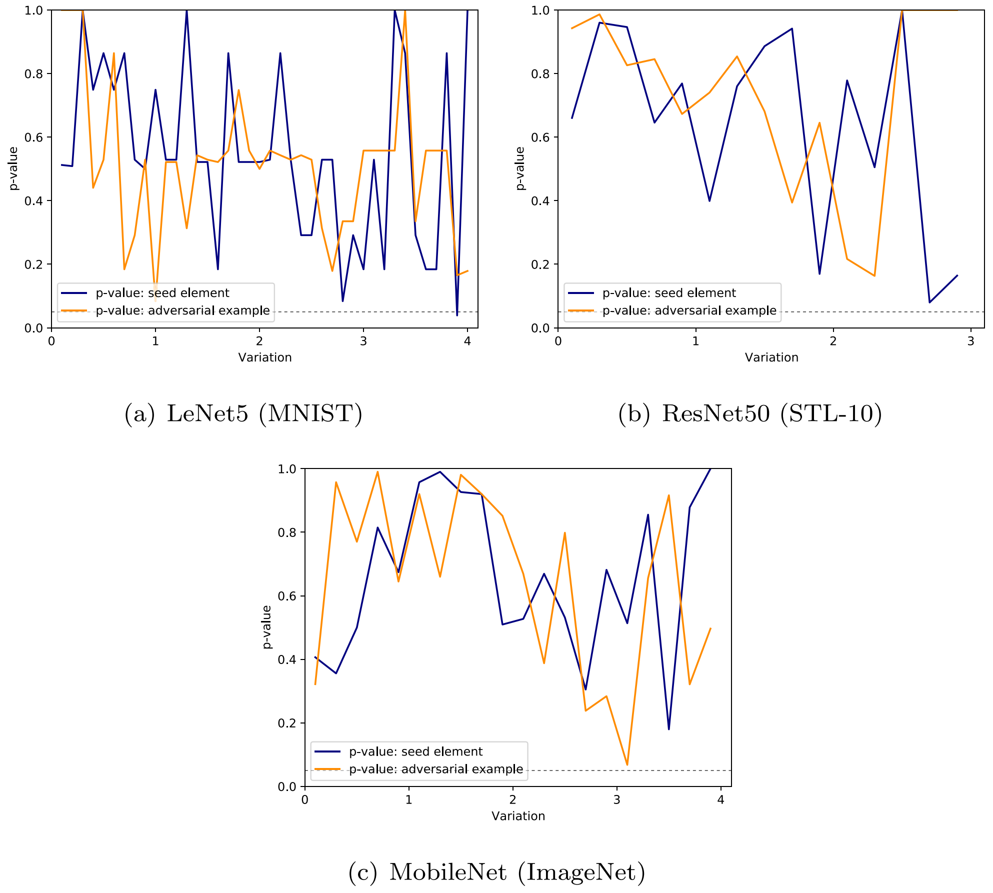


Fig. 6. The statistical result of Mann Whitney U-test on mutated images discovered by mixed and constrained mutation. We performed separate tests on seed elements and adversarial examples of one variation section, respectively. The p-values of seed elements are plotted by navy line, and the p-values of adversarial examples are plotted by orange line. The horizontal dashed line indicates the significance level, $\alpha = 0.05$.

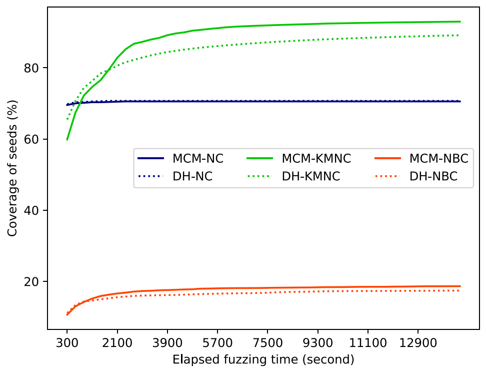
on STL-10, it yielded 14.53%, 99.84%, and 33.03% of coverage rate with NC, KMNC, and NBC, respectively. The gap in coverage rate between two fuzzers became larger on ImageNet. MCM achieved 24.95%, 92.32%, and 45.75% of coverage rate with NC, KMNC, and NBC, while DeepHunter achieved 21.08%, 89.86%, and 43.45%, respectively. In addition, MCM increased the coverage rate of NC by 2.39%p, 3.7%p, and 5.36%p, on other ImageNet models (i.e., Inception-v3, ResNet50, and ViT-L/32), than DeepHunter which yielded 11.36%, 8.3%, and 70.87% of coverage rate.

The fuzzer provides more opportunities to detect diverse vulnerabilities when there are more seeds and the coverage is higher. In reality, the coverage of adversarial examples was similar to that of the seed elements. In addition, the gap in the coverage of the adversarial examples between the two fuzzers was slightly larger than that of the seed elements. NC saturated early in LeNet5 and ViT-L/32 because these models use few neurons to measure NC. Nevertheless, the NC increased continuously in the other model architectures.

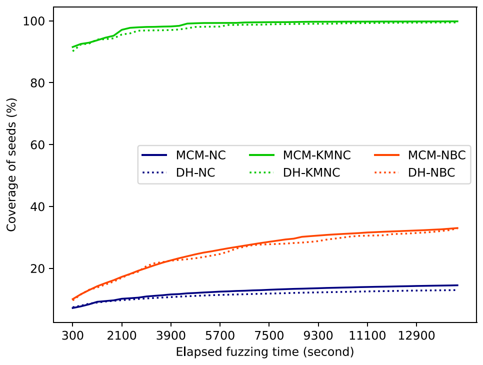
4.2.3. RQ3. Vulnerability Detection

Although the combination of multiple image transformation algorithms was effective for discovering informative seed elements, its effectiveness can be emphasized more in vulnerability detection. Fig. 8 depicts adversarial examples detected by the MCM. They visually preserve the semantics of their original images but cause misclassification toward other classes. Since each of the examples is critical for the model, the fuzzer should detect them as many and diverse as possible. MCM was more effective than DeepHunter at detecting the adversarial examples.

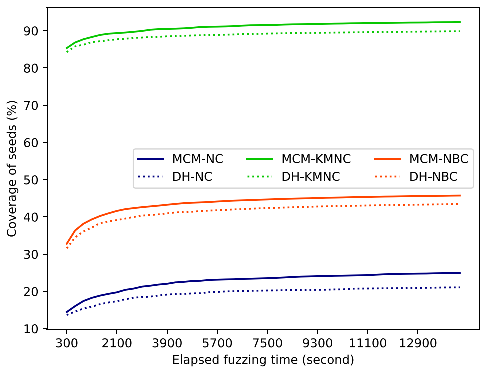
Quantity of adversarial examples. Table 5 presents the number of unique adversarial examples after the completion of the fuzzing execution. Even on the same target model, different coverage criteria detected a largely different quantities of adversarial examples. Fuzzers using MCM detected more adversarial examples than DeepHunter in all environments, except the NC of LeNet5. Because the NC of LeNet5 cannot discover new seeds, it operates similarly to random testing. Consequently, we cannot expect the efficacy of coverage and mutation in this environment. MCM and DeepHunter commonly detected more adversarial examples on MNIST than STL-10 and ImageNet, because the parameter size of LeNet5 is small. For our best case, the difference in the number of adversarial examples between MCM and DeepHunter was 321% for the



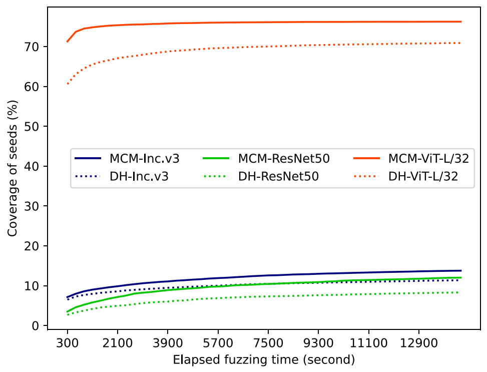
(a) LeNet5 (MNIST)



(b) ResNet50 (STL-10)



(c) MobileNet (ImageNet)



(d) Inception-v3, ResNet50, and ViT-L/32 (ImageNet, NC only)

Fig. 7. Averaged coverage (%) of seed elements. MCM and DH in the legend denote the approach proposed in this study and DeepHunter, respectively. Although the achieved coverage differs among different coverage criteria, our approach always covers more diverse neuron activations than DeepHunter for the same coverage criterion. The coverage of adversarial examples appeared to be similar to the coverage of seed elements.

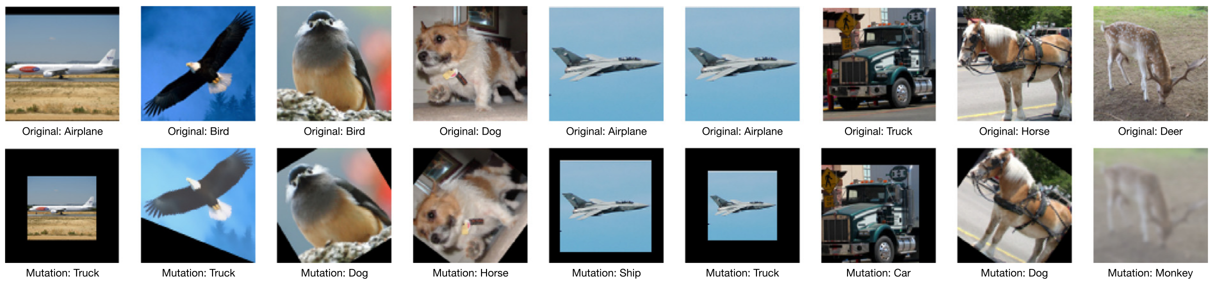


Fig. 8. Erroneous images for ResNet50 on STL-10 detected by the approach proposed in this study. The upper images are original images and the lower images their mutation results. The mutated images caused misclassifications of the target model to other classes. The predicted class of an image is described below the image. With different mutation settings, the same original image can be misclassified to different target classes.

KMNC of LeNet5. For that case, DeepHunter detected 22,628 examples and the MCM detected 95,219.8 examples. We note that their difference in the number of seed elements was 12% for that case. MCM detected 35% more adversarial examples than DeepHunter, even for the NC of ResNet50 on STL-10, which is the worst case. MCM detected more than twice adversarial examples compared to DeepHunter for all cases on ImageNet. For instance, DeepHunter detected 6,078.6 adversarial examples with NBC on MobileNet, but MCM detected 19,273.8 examples, showing 217% of difference. DeepHunter could not detect adversarial examples of more than 10,000 on ImageNet models such as Inception-v3, ResNet50, and ViT-L/32. However, MCM detected 14,362.6 and 26.737 adversarial examples on ResNet50 of ImageNet and ViT-L/32, respectively. The

Table 5
The number of unique adversarial examples that triggered misclassification of the model. The results were averaged over the five fuzzing executions. The column MCM is the approach proposed in this study. DH denotes DeepHunter. LeNet5 was tested on the MNIST dataset, and ResNet50 was tested on the STL-10 and ImageNet datasets. The other models were tested on the ImageNet dataset. LeNet5, ResNet50 for STL-10, and MobileNet were tested using NC, KMNC, and NBC. Inception-v3, ResNet50 for ImageNet, and ViT-L/32 were tested using only the neuron coverage.

Coverage	**LeNet5**		**ResNet50 (STL-10)**		**MobileNet**		**Inception-v3**		**ResNet50 (ImageNet)**		**ViT-L/32**	
Criteria	**MCM**	**DH**	**MCM**	**DH**	**MCM**	**DH**	**MCM**	**DH**	**MCM**	**DH**	**MCM**	**DH**
NC	5,692.6	8,638.8	17,222.0	12,721.2	22,799.6	12,521.0	7,398.2	2,421.4	14,362.6	6,832.0	26,737.2	9,055.8
KMNC	95,219.8	22,628.8	9,729.4	6,286.2	15,163.6	5,508.0	N/A	N/A	N/A	N/A	N/A	N/A
NBC	22,335.6	14,018.0	12,685.2	4,847.8	19,273.8	6,078.6	N/A	N/A	N/A	N/A	N/A	N/A

average difference in the number of unique adversarial examples across all environments was 132%. This indicates that MCM detects more individual images such as Fig. 8, regardless of the dataset, target model, and coverage criterion.

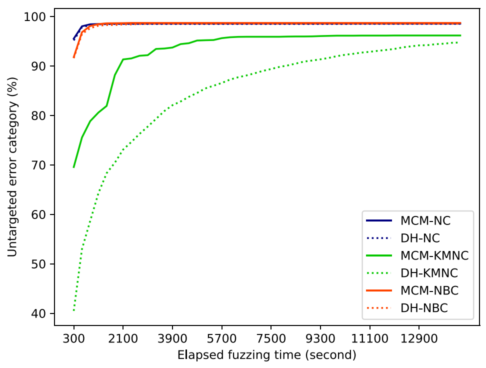
The performance of MCM was always better in terms of coverage of adversarial examples. The result was almost the same as that in Fig. 7, so we omit the graph for the coverage of adversarial examples. Nevertheless, the gap in coverage between the two approaches increased. Higher coverage of adversarial examples means that MCM detects vulnerabilities from more diverse neuron activations.

Diversity of adversarial examples. In addition to the perspective of adversarial examples as the end-point error, we identified the number of diverse errors that can occur from the original images that comprise the initial seed corpus. This evaluation approach was introduced by DeepHunter [11]. To measure the diversity of adversarial examples, DeepHunter defined an error category identical to the number of original images that led to adversarial examples. However, as shown in Fig. 9, it is easy to trigger adversarial examples in most of the original images, regardless of the fuzzing scheme. Therefore, we cannot clearly identify the difference in performance using only a single definition of the error category. We have redefined the error category for more specific comparison. The original error category of DeepHunter is denoted as the untargeted error category. We further define the targeted error category by considering a case in which adversarial examples in an original image are misclassified into multiple classes. One targeted error category is a pair consisting of an original image and a misclassified class. That is, the maximum number of targeted error categories is identical to (the number of original images $\times$ (the number of total classes - 1)).

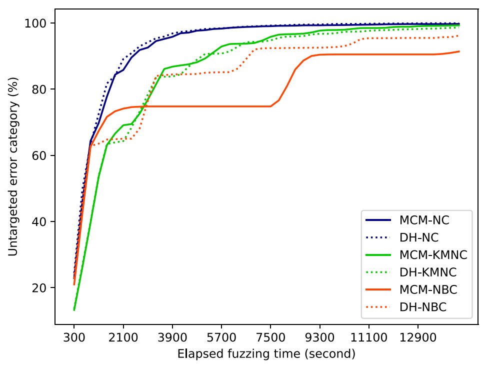
Fig. 9 depicts the ratio of untargeted error categories during the fuzzing execution. All fuzzing executions resulted in almost 100% of untargeted error categories at the end. The number of untargeted error categories for MCM and DeepHunter was the same for NC of MobileNet. The difference in untargeted error categories between fuzzers became larger for models that used only NC (i.e., Inception-v3, ResNet50 of ImageNet, and ViT-L/32). For instance, MCM discovered 91.6% of untargeted error categories on Inception-v3, which is 6%p larger than DeepHunter, showing the greatest gap among all environments. MCM discovered slightly better performance than DeepHunter for the other environments because DeepHunter already achieved a large percentage of untargeted error categories for them. For ResNet50 on STL-10 and MobileNet, there was some point of time that DeepHunter achieved more untargeted error categories in the middle of execution but the performance of MCM was better at the end. This indicates that overlapping multiple transformations gets more effective as the fuzzing proceeds. For the other models, MCM maintained more untargeted error categories throughout the execution. Exceptionally, DeepHunter discovered 96.18% of untargeted error categories but MCM discovered only 91.38% with 4.8% of reduction. One plausible reason is that MCM focuses only on images that are easier to increase coverage rather than traversing diverse original images. We can see from the graph that MCM got stuck twice during the fuzzing execution. To solve this problem, we need to regulate the popularity of each original image in seed selection. We leave the advanced seed selection algorithm as future work.

Fig. 10 shows the ratio of targeted error categories during the fuzzing execution. It is more difficult to target a certain class or all classes than simply causing a misclassification from one original image. In this respect, targeted error categories are more rarely discovered than untargeted error categories. The ratio of targeted error categories was relatively higher on MNIST and STL-10 than ImageNet since MNIST and STL-10 have only ten classes. MCM with KMNC discovered 51.6% of targeted error categories on LeNet5, which was 26.46%p higher than DeepHunter with KMNC, while the other coverage criteria on LeNet5 achieved targeted error categories under 25%. The targeted error categories of MCM with KMNC are equal to 4,644 pairs of an original image and a misclassified class. This means that MCM detected about 4.5 vulnerable classes for each original image. Since NBC and NC cannot affect the fuzzer largely on LeNet5, MCM discovered slightly more targeted error categories than DeepHunter for those coverage criteria. The MCM discovered 23.31% of targeted error categories with NC, and DeepHunter discovered 23.11% of targeted error categories. Ratios of targeted error categories for NBC were 24.72% and 23.73% for the MCM and DeepHunter, respectively.

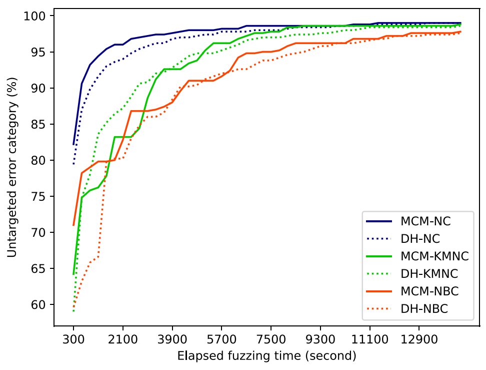
Although the MCM with NBC discovered less quantity of untargeted error categories than DeepHunter on ResNet50 of STL-10, it showed better performance in terms of targeted error categories. While DeepHunter discovered 27.84% of targeted error categories with NBC, the MCM discovered 29.80% of targeted error categories which is about 2.0%p larger. For KMNC, the MCM discovered 37.4% of targeted error categories but DeepHunter discovered 33.67% which is 3.73%p lower. The MCM



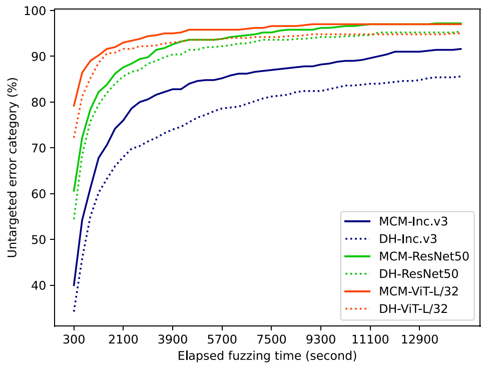
(a) LeNet5 (MNIST)



(b) ResNet50 (STL-10)



(c) MobileNet (ImageNet)



(d) Inception-v3, ResNet50, and ViT-L/32 (ImageNet, NC only)

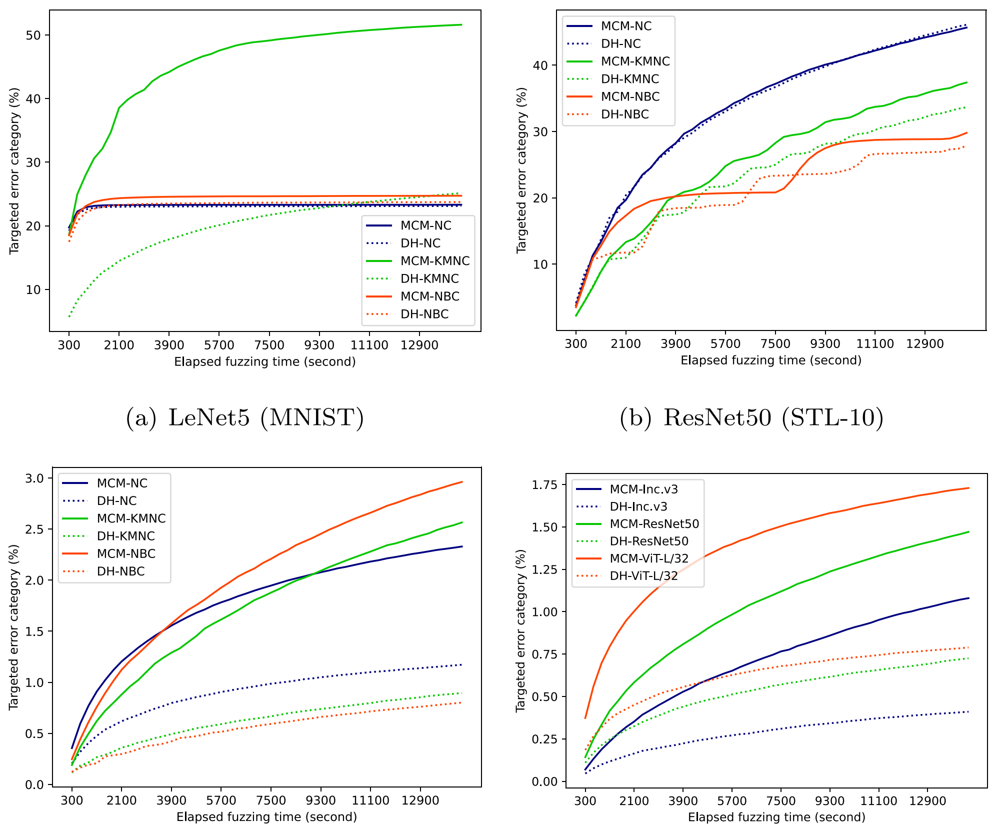
Fig. 9. Averaged percentage of untargeted error categories (%). MCM and DH in the legend denote the approach proposed in this study and DeepHunter, respectively. The untargeted error category is identical to the number of initial seeds, which eventually leads to an adversarial example. Hence, the maximal number of untargeted error categories was 1,000 for LeNet5 and STL-10 and 100 for the other models.

was worse than DeepHunter with NC, but the difference was only 0.43%p. Ratios of targeted error categories were 45.68% and 46.10% for the MCM and DeepHunter, respectively.

The ratio of targeted error categories is more difficult to increase on ImageNet because the maximum number of targeted error categories is enormous. Nevertheless, the difference in the fuzzing performance of both mutation schemes increased compared to that of MNIST and STL-10. MCM with NBC achieved 2.96% of the targeted error categories, which is 2.7 times higher than that of DeepHunter on MobileNet. Although this ratio still seems numerically trivial, it is identical to the 2,957 pairs of original images and misclassified classes. This also indicates that the MCM with NBC detected errors in approximately 30 classes for each original image. The MCM also discovered 2.33% and 2.56% of targeted error categories with NC and KMNC on MobileNet, respectively. DeepHunter discovered targeted error categories of less than 1% (0.9% with KMNC and 0.8% with NBC), except 1.17% with NC on MobileNet. In addition, the MCM discovered 1.47% and 1.73% of targeted error categories on ResNet50 and ViT-L/32 while DeepHunter discovered 0.72% and 0.79%, respectively. Even for the worst case, the MCM discovered 1.08% of the targeted error categories on Inception-v3, which was 2.6 times as high as 0.41% of DeepHunter. This implies that the MCM can detect errors in approximately 10 classes for each original image, even in the worst case. DeepHunter became slower to discover targeted error categories as fuzzing proceeds for all cases on ImageNet, but the MCM still showed its detection power at the end of fuzzing. Therefore, we expect that the MCM can achieve a much higher ratio of targeted error categories with more fuzzing budget.

5. Discussion

In this study, we propose a new mutation approach for coverage-guided deep learning fuzzing. Although we used the same image transformation algorithms as DeepTest [12] and DeepHunter [11], we generated only valid inputs that preserved the semantics of their original inputs while combining multiple transformation algorithms. Moreover, a fuzzer using our



(a) LeNet5 (MNIST)　(b) ResNet50 (STL-10)

(c) MobileNet (ImageNet)　(d) Inception-v3, ResNet50, and ViT-L/32 (ImageNet, NC only)

Fig. 10. Averaged percentage of targeted error categories (%). MCM and DH in the legend denote the approach proposed in this study and DeepHunter, respectively. The targeted error category is identical to the number of pairs (initial seed, misclassified class). Hence, the maximal number of targeted error categories was 9,000 for LeNet5 and STL-10 and 99,900 for the other models.

mutation strategy can discover broader coverage and more diverse errors by generating various mutants. However, there are several issues concerning threats to the validity and future work of this study.

Threats to the validity. Some might be concerned about whether MCM mutants preserve the semantics of their original input. The reliability of the mutation is the primary objective of fuzzing. Therefore, we did not attempt to violate this issue. While previous studies defined the parameters of transformation algorithms on their own, we defined reliable transformation parameters based on the user study and statistical test. Furthermore, we conservatively set the permission range combination of transformations by defining a low value of the constraint C. If we alleviate the constraint value, the distortion increases, but the defect-detection performance also increases. In the verification of the feasibility of the mutation results, we utilized a statistical test in which samples were collected by asking for one image for multiple participants. On the other hand, DeepHunter asked one image for a single participant and reported only the accuracy of the images. Eventually, we statistically determined that our mutation results preserve the semantics of the original inputs.

Our experiments were performed only on MNIST and ImageNet. However, constraints of MCM are dependent on corresponding datasets. Because our mutation strategy is accompanied by a user study, we had difficulty reporting other datasets. Nevertheless, we focused on the definition of reliable transformation parameters rather than verifying a diverse dataset. Furthermore, if statistical tests are supported, the validity of transformation parameters can be verified on a new dataset without using whole images. MNIST, STL-10, and ImageNet, which were selected for this study, represent grayscale and colored images. We expect that fuzzer using MCM also performs better than existing fuzzers on a new dataset because MCM showed the best performance in our experiments for all evaluation factors.

The coverage criteria that we used in this study were designed earlier. Those criteria are difficult to apply to models with different structures. The coverage criteria of DeepXplore [9] and DeepGauge [10] can be applied to fully connected and convolutional layers. The ViT-L/32 model evaluated in this study deploys a new type of layer, the transformer encoder [21,20]. Consequently, the coverage of this model was saturated early. Therefore, new criteria are required for measuring the coverage of the transformer layer. Likewise, new coverage criteria should be designed considering the structural differences in network architectures applied to other domains. For instance, DeepStellar designed coverage criteria to observe the change in the states of a hidden layer for recurrent neural networks [22].

Future work. We used as many transformation algorithms as possible to verify image recognition systems. In addition to the affine, linear, and convolution transformations, we can further consider other transformation algorithms, such as JPEG compression, image quilting, and blending [23]. Changes in light, rain, and fog can be also considered. However, the mentioned algorithms do not have specific criteria to measure the variation; therefore, we cannot apply the constraint to these algorithms. Nevertheless, we can use the aforementioned algorithms as candidates for mutation. One of the solutions is to restrict the usage number of algorithms, as we used blur only once for an original image. In addition, Li et al. proposed a non-parametric mutation that employs a generative adversarial network [24]. As their approach is also orthogonal to our approach, we can combine both mutation algorithms to simulate more diverse situations.

As mentioned above, parameter definition solely based on a user study with human intelligence improves the reliability of mutation but hinders its scalability. An automated approach can be one of the solutions to efficiently define transformation parameters on other image datasets. For instance, because affine transformation changes the position of an object in images, we can verify whether the bounding box is still in the image after the mutation. Some datasets such as ImageNet provides bounding boxes of images for the label of object detection.

Although the MCM can be transferred to other image datasets, another consideration is fuzzing other domains. To test other domains, the fuzzer should change its modules such as mutation and coverage. We can apply our mutation methodology to other domains even though the data of the domains have different properties from the image dataset. As we measured the variation of each image transformation algorithm for constraint, we can define parameters and measure variations for transformation algorithms of other domains. In natural language processing (NLP), we can change the location of characters or words of a text sentence through affine transformations or change the value of characters or words using linear transformations. In this case, a possible constraint is verifying the grammar of the mutated sentence (e.g., homonyms and homographs are allowed when changing words) or the similarity between the sentence and its previous state.

6. Related work

6.1. Testing deep learning based systems

The decision of a deep learning-based system is hard to comprehend by a human because of the enormous parameter space of the model. Previous studies that considered the perspective in traditional software testing have been proposed to facilitate the verification of deep learning systems in various corner-case situations. Reluplex [25] attempted to verify the lower bound of vulnerabilities of a deep learning system. It calculates the range of certain inputs that the system never malfunctions using a constraint solver. However, Reluplex has a limitation in that it cannot be applied to models that have a large parameter space and use an activation function instead of ReLU.

In contrast to Reluplex, some previous studies have attempted to verify the upper bound of the model by discovering inputs that trigger mispredictions. In this context, coverage criteria for deep learning models are utilized to measure the scalability of a test suite in systematic testing or to generate interesting test cases in coverage-guided fuzzing. Various approaches have been proposed, in addition to the coverage criteria mentioned in Section 2.1 [9,10]. MC/DC coverage measures whether the sign or value of neuron activation has been changed by a new input compared with the previous input [26]. Surprise adequacy measures the likelihood distance of new input to its predicted class compared with other classes [27]. The surprise adequacy of the new input increases when the input is closer to classes other than the original. TensorFuzz converted the concept of neuron coverage to a distance-based approach. It added a new seed element whose distance to its nearest neighbor is above a certain threshold [14]. Although these coverage criteria are well-defined, we adopted the coverage criteria of DeepXplore [9] and DeepGauge [10], which are efficient for quantifying neuron activations and are popularly utilized.

6.2. Mutation strategies for deep learning models

Even a well-trained deep learning model raises misbehavior by changing the inputs [28,15,29]. Security analysts for deep learning models attempt to identify erroneous inputs and then cope with them. In this respect, adversarial attacks or image transformation algorithms can be applied to a given dataset.

Adversarial attacks are the most representative method for triggering the misbehavior of a model. They generated synthetic images by adding perturbations that are imperceptible to humans to the original inputs. The magnitude of the perturbations was restricted by the l_p-norm. White-box attacks, where knowledge such as the parameters of the target model is given, generate perturbations based on the loss function and gradient [30–34]. Black-box attacks estimate the gradient through multiple queries and use the gradient as a perturbation [35–38]. Therefore, the input transformation of adversarial attacks is far from the image transformation algorithms used in this study. Some studies have utilized image resizing [39], image translation [40], and random pixel dropping [41] to improve the transferability of attacks on black-box models. However, they only used image transformation algorithms to generate the gradient, and the transformation algorithms were not applied to the resulting images.

White-box testing of deep-learning systems is the closest research area to our work. In this regard, DeepXplore [9] defined an objective function to increase the neuron coverage and to trigger misbehavior of the model, and then applied gradient ascent to the input to maximize the objective function. DLFuzz [13] uses an objective function similar to DeepXplore, but results in small perturbations. DLFuzz adopted a coverage-guided approach, in which only mutants with new neuron coverage can proceed to the next state. While the mutations in DeepXplore and DLFuzz are influenced by the feedback of the objective function, TensorFuzz [14], DeepTest [12], and DeepHunter [11] apply fully random transformations. The mutation strategies of the latter three studies are freer, but their coverage-guided approach makes the fuzzer focus on fresh inputs. Although these studies are closest to our work, they covered only a small part of our mutation strategy. DeepTest permits the overlap of multiple image transformation algorithms, but it does not consider random noise or constraints for them. TensorFuzz uses random noises within l_∞-norm constraint. DeepHunter advances the mutation strategy of DeepTest and TensorFuzz. It applies with l_0 and l_∞ constraint for random noises. In addition, it allows one image transformation algorithm to be applied only once to the original image for the input validity. However, DeepHunter may still harm the semantics of original images, and its limited constraint hinders diverse inputs. We tried to improve the validity of the mutated inputs by applying the user study-based constraint to image transformations while permitting the overlap of multiple transformation algorithms for the input diversity.

Deep learning model fuzzers have been developed after DeepHunter. RobOT [42] applied the loss-based approach, rather than coverage criteria, to fuzzer. It measured the importance of test cases during fuzzing, and then it retrained the model using effective test cases to improve the robustness of the model. Sun et al. [43] retrained the model by coverage-guided fuzzing. They newly applied the independence neuron coverage, which investigates the interaction between neurons, to fuzzer. These two recent fuzzers adopt the mutation approach of DeepHunter. Therefore, they can be further improved by applying our mutation strategy, preserving their other fuzzing components.

Some fuzzers utilize a genetic algorithm to generate interesting test cases [44,45]. Gao et al. augmented the training dataset by applying a genetic algorithm to the input mutations [44]. They created parameter sets for all image transformation algorithms. For each fuzzing iteration, they substituted a part of the current parameter set with a part of a more effective parameter set among the other sets. They used the loss function rather than coverage criteria to measure the effectiveness of the parameter set. Although they combine all transformation algorithms, they have a pitfall in generating invalid inputs because their parameter ranges are subjective, and they do not consider the constraint.

7. Conclusion

In this paper, we proposed an input mutation strategy called the mixed and constrained mutation for coverage-guided fuzzing in deep learning-based systems. We identified that mutation strategies of previous fuzzing schemes generate very limited mutation results or images that deviate from the human perception, which degrades the fuzzing performance. To solve this problem, we designed the mutation approach considering two prime objectives corresponding to input validity and diversity. The diversity of the inputs was achieved by combining multiple image transformation algorithms. The validity of the inputs was accomplished by imposing the constraint on transformation algorithms. In particular, to improve the reliability of the constraint, relative variations of transformation algorithms are measured with transformation parameters defined by user study and statistical tests. In the evaluation, our mutation strategy enhanced the fuzzing performance significantly in terms of coverage and vulnerability detection by virtue of diverse inputs with a high mutation success rate. Although our work targets two datasets in the image domain, we expect our approach to be extended to fuzzing other image datasets or other domains.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2019R1A2C1088802) and this research was supported by Institute for Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government (MSIT) (No.2018–0-00513, Machine Learning Based Automation of Vulnerability Detection on Unix-based Kernel). The authors thank Prof. Lei Ma for sharing his DeepHunter code to foster this study.

References

[1] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv:1409.1556.

[2] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 1–9.
[3] G. Hinton, L. Deng, D. Yu, G.E. Dahl, A.-R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T.N. Sainath, et al, Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups, IEEE Signal Processing Magazine 29 (6) (2012) 82–97.
[4] V. Gulshan, L. Peng, M. Coram, M.C. Stumpe, D. Wu, A. Narayanaswamy, S. Venugopalan, K. Widner, T. Madams, J. Cuadros, et al, Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs, Jama 316 (22) (2016) 2402–2410.
[5] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L.D. Jackel, M. Monfort, U. Muller, J. Zhang, et al., End to end learning for self-driving cars, arXiv preprint arXiv:1604.07316.
[6] K. Grosse, N. Papernot, P. Manoharan, M. Backes, P. McDaniel, Adversarial perturbations against deep neural networks for malware classification, arXiv preprint arXiv:1606.04435.
[7] M. Zalewski, Americal fuzzy lop (2014). URL: https://lcamtuf.coredump.cx/afl/.
[8] K. Serebryany, LibFuzzer - a library for coverage-guided fuzz testing, LLVM project.
[9] K. Pei, Y. Cao, J. Yang, S. Jana, DeepXplore: Automated whitebox testing of deep learning systems, in: Proceedings of the ACM Symposium on Operating Systems Principles (SOSP), ACM, 2017, pp. 1–18.
[10] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu, J. Zhao, Y. Wang, DeepGauge: Multi-granularity testing criteria for deep learning systems, in: Proceedings of the ACM/IEEE International Conference on Automated Software Engineering (ASE), ACM, 2018, pp. 120–131.
[11] X. Xie, L. Ma, F. Juefei-Xu, M. Xue, H. Chen, Y. Liu, J. Zhao, B. Li, J. Yin, S. See, DeepHunter: A coverage-guided fuzz testing framework for deep neural networks, in: Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), ACM, 2019, pp. 146–157.
[12] Y. Tian, K. Pei, S. Jana, B. Ray, DeepTest: Automated testing of deep-neural-network-driven autonomous cars, in: Proceedings of the International Conference on Software Engineering (ICSE), 2018, pp. 303–314.
[13] J. Guo, Y. Jiang, Y. Zhao, Q. Chen, J. Sun, DLFuzz: Differential fuzzing testing of deep learning systems, in: Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), ACM, 2018, pp. 739–743.
[14] A. Odena, C. Olsson, D.G. Andersen, G. Ian, TensorFuzz: Debugging neural networks with coverage-guided fuzzing, in: Proceedings of the International Conference on Machine Learning (ICML), PMLR, 2019, p. 4901–4911.
[15] I.J. Goodfellow, J. Shlens, C. Szegedy, Explaining and harnessing adversarial examples, in: Proceedings of the International Conference on Learning Representations (ICLR), 2015.
[16] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.
[17] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778.
[18] A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, Mobilenets: Efficient convolutional neural networks for mobile vision applications, arXiv preprint arXiv:1704.04861.
[19] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the inception architecture for computer vision, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 2818–2826.
[20] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al., An image is worth 16x16 words: Transformers for image recognition at scale, in: Proceedings of the International Conference on Learning Representations (ICLR), 2021.
[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), 2017, pp. 6000–6010.
[22] X. Du, X. Xie, Y. Li, L. Ma, Y. Liu, J. Zhao, Deepstellar: Model-based quantitative analysis of stateful deep learning systems, in: Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), 2019, pp. 477–487.
[23] C. Guo, M. Rana, M. Cisse, L. van der Maaten, Countering adversarial images using input transformations, in: International Conference on Learning Representations (ICLR), 2018.
[24] Z. Li, M. Pan, T. Zhang, X. Li, Testing dnn-based autonomous driving systems under critical environmental conditions, in: Proceedings of International Conference on Machine Learning (ICML), PMLR, 2021, pp. 6471–6482.
[25] G. Katz, C. Barrett, D.L. Dill, K. Julian, M.J. Kochenderfer, Reluplex: An efficient smt solver for verifying deep neural networks, in: Proceedings of the International Conference on Computer Aided Verification (CAV), Springer, 2017, pp. 97–117.
[26] Y. Sun, X. Huang, D. Kroening, J. Sharp, M. Hill, R. Ashmore, Testing deep neural networks, arXiv preprint arXiv:1803.04792.
[27] J. Kim, R. Feldt, S. Yoo, Guiding deep learning system testing using surprise adequacy, in: Proceedings of the IEEE/ACM International Conference on Software Engineering (ICSE), IEEE, 2019, pp. 1039–1049.
[28] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus, Intriguing properties of neural networks, in: Proceedings of the International Conference on Learning Representations (ICLR), 2014.
[29] A. Nguyen, J. Yosinski, J. Clune, Deep neural networks are easily fooled: High confidence predictions for unrecognizable images, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2015, pp. 427–436.
[30] N. Carlini, D. Wagner, Towards evaluating the robustness of neural networks, in: Proceedings of the IEEE Symposium on Security and Privacy (SP), IEEE, 2017, pp. 39–57.
[31] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z.B. Celik, A. Swami, The limitations of deep learning in adversarial settings, in: Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P), IEEE, 2016, pp. 372–387.
[32] A. Kurakin, I. Goodfellow, S. Bengio, Adversarial examples in the physical world, in: Proceedings of the International Conference on Learning Representations (ICLR), 2017.
[33] J. Rony, L.G. Hafemann, L.S. Oliveira, I.B. Ayed, R. Sabourin, E. Granger, Decoupling direction and norm for efficient gradient-based l2 adversarial attacks and defenses, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2019, pp. 4322–4330.
[34] K. Xu, S. Liu, P. Zhao, P.-Y. Chen, H. Zhang, Q. Fan, D. Erdogmus, Y. Wang, X. Lin, Structured adversarial attack: Towards general implementation and better interpretability, in: Proceedings of the International Conference on Learning Representations (ICLR), 2019.
[35] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z.B. Celik, A. Swami, Practical black-box attacks against machine learning, in: Proceedings of the ACM on Asia Conference on Computer and Communications Security (AsiaCCS), 2017, pp. 506–519.
[36] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, C.-J. Hsieh, ZOO: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models, in: Proceedings of the ACM Workshop on Artificial Intelligence and Security (AISec), 2017, pp. 15–26.
[37] Y. Li, L. Li, L. Wang, T. Zhang, B. Gong, NAttack: Learning the distributions of adversarial examples for an improved black-box attack on deep neural networks, in: Proceedings of the International Conference on Machine Learning (ICML), PMLR, 2019, pp. 3866–3876.
[38] J. Chen, M.I. Jordan, M.J. Wainwright, Hopskipjumpattack: A query-efficient decision-based attack, in: Proceedings of the IEEE symposium on security and privacy (SP), IEEE, 2020, pp. 1277–1294.
[39] C. Xie, Z. Zhang, Y. Zhou, S. Bai, J. Wang, Z. Ren, A.L. Yuille, Improving transferability of adversarial examples with input diversity, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2019, pp. 2730–2739.
[40] Y. Dong, T. Pang, H. Su, J. Zhu, Evading defenses to transferable adversarial examples by translation-invariant attacks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 4312–4321.
[41] Z. Wang, H. Guo, Z. Zhang, W. Liu, Z. Qin, K. Ren, Feature importance-aware transferable adversarial attacks, in: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 2021, pp. 7639–7648.
[42] J. Wang, J. Chen, Y. Sun, X. Ma, D. Wang, J. Sun, P. Cheng, Robot: Robustness-oriented testing for deep learning systems, in: Proceedings of IEEE/ACM International Conference on Software Engineering (ICSE), IEEE, 2021, pp. 300–311.

[43] W. Sun, Y. Lu, M. Sun, Are coverage criteria meaningful metrics for dnns?, in: Proceedings of International Joint Conference on Neural Networks (IJCNN), IEEE, 2021, pp. 1–8.
[44] X. Gao, R.K. Saha, M.R. Prasad, A. Roychoudhury, Fuzz testing based data augmentation to improve robustness of deep neural networks, in: Proceedings of the IEEE/ACM International Conference on Software Engineering (ICSE), IEEE, 2020.
[45] V. Riccio, N. Humbatova, G. Jahangirova, P. Tonella, Deepmetis: Augmenting a deep learning test set to increase its mutation score, in: Proceedings of IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, 2021, pp. 355–367.