DeepHunter: A Coverage-Guided Fuzz Testing Framework for Deep Neural Networks

Xiaofei Xie
Nanyang Technological University
Singapore

Lei Ma[*]
Kyushu University
Japan

Felix Juefei-Xu
Carnegie Mellon University
USA

Minhui Xue
The University of Adelaide
Australia

Hongxu Chen
Nanyang Technological University
Singapore

Yang Liu
Nanyang Technological University
Zhejiang Sci-Tech University, China

Jianjun Zhao
Kyushu University
Japan

Bo Li
University of Illinois at
Urbana-Champaign, USA

Jianxiong Yin
Simon See
NVIDIA AI Tech Centre, Singapore

ABSTRACT

The past decade has seen the great potential of applying deep neural network (DNN) based software to safety-critical scenarios, such as autonomous driving. Similar to traditional software, DNNs could exhibit incorrect behaviors, caused by hidden defects, leading to severe accidents and losses. In this paper, we propose *DeepHunter*, a coverage-guided fuzz testing framework for detecting potential defects of general-purpose DNNs. To this end, we first propose a metamorphic mutation strategy to generate new semantically preserved tests, and leverage multiple extensible coverage criteria as feedback to guide the test generation. We further propose a seed selection strategy that combines both diversity-based and recency-based seed selection. We implement and incorporate 5 existing testing criteria and 4 seed selection strategies in *DeepHunter*. Large-scale experiments demonstrate that (1) our metamorphic mutation strategy is useful to generate new valid tests with the same semantics as the original seed, by up to a 98% validity ratio; (2) the diversity-based seed selection generally weighs more than recency-based seed selection in boosting the coverage and in detecting defects; (3) *DeepHunter* outperforms the state of the arts by coverage as well as the quantity and diversity of defects identified; (4) guided by corner-region based criteria, *DeepHunter* is useful to capture defects during the DNN quantization for platform migration.

CCS CONCEPTS

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

KEYWORDS

Deep learning testing, metamorphic testing, coverage-guided fuzzing

ACM Reference Format:
Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. 2019. *DeepHunter*: A Coverage-Guided Fuzz Testing Framework for Deep Neural Networks. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '19), July 15–19, 2019, Beijing, China.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3293882.3330579

1 INTRODUCTION

Over the past decade, deep learning (DL) has achieved great success in many cutting-edge intelligent applications, such as image processing [12], speech recognition [5, 20], video, and board games [34, 42]. However, like traditional software, DNN-based software could still have defects, in spite of having achieved very high testing accuracy as demonstrated by adversarial attacks [7]. Some real-world cases, such as the Tesla/Uber accident [46] and intelligent audiobots (*e.g.*, Siri, Alexa) manipulated with hidden command [3], have been reported and they will potentially cause severe safety and security problems. The importance of quality and security assurance of DL starts to draw attention, especially for those applied in safety- and mission-critical scenarios.

Like traditional software, DNN-based software also needs to be systematically tested before the deployment. The systematic testing of a DNN model to identify its potential defects and vulnerability at an early stage is of great importance. More diverse defects provide feedback to a DL developer for further root-cause analysis, robustness enhancement, *etc*. For traditional software, testing is a well-established and commonly used technique to perform quantitative analysis on software quality [6]. However, the testing technique of traditional software could not be directly applied to DNN-based software due to the fundamental difference in the programming paradigm and the development process (Fig. 1), and thus making testing of the DL software a new challenge.

Some recent progress has been made on testing DL software, which are centered around two important issues in software testing: 1) testing criteria, *e.g.*, Neuron Coverage in [36], *k*-multisection Neuron Coverage (KMNC) and Neuron Boundary Coverage (NBC)

[*] Lei Ma is the corresponding author of this paper, E-mail:malei@ait.kyushu-u.ac.jp.

ISSTA '19, July 15–19, 2019, Beijing, China
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6224-5/19/07…$15.00
https://doi.org/10.1145/3293882.3330579

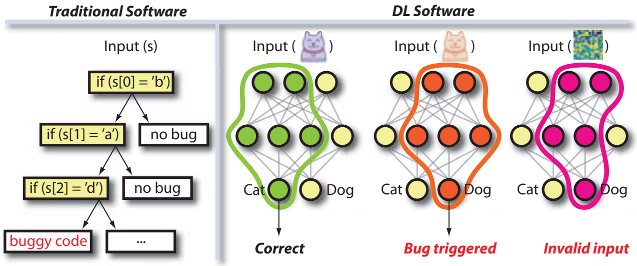

Figure 1: Decision flow comparison between the traditional (L) and the DNN (R) software. We highlight the neurons that make major contributions to the final output decision.

in [31]; 2) testing strategy to maximize the coverage, *e.g.*, the coverage-guided fuzz (CGF) testing technique in [35, 44]. However, for the new problem (*i.e.*, DNN testing), the current research is still at an early stage. There lacks a comparative study on the effectiveness of different testing criteria and testing strategies, leaving many challenges and open questions unresolved, such as whether the existing proposed testing criteria are indeed useful and whether existing testing strategies in traditional software are still effective.

A fundamental challenge, regarding the usefulness of proposed testing criteria, comes from the decision logic representation (*i.e.*, a DL encodes logic as a DNN). The testing criteria of traditional software (*e.g.*, line coverage, branch coverage) directly follow the source code structure, but such explicit structures vanish for existing testing criteria (*e.g.*, neuron coverage, KMNC) of DNNs. Intuitively, increasing the test criteria to cover the specific code fragments (*e.g.*, branches) is a necessity condition to detect defects (see Fig. 1). Such a casual linkage becomes blurred when it comes to DL testing criteria. It is not obvious that covering these testing criteria of DL would still facilitate the defect detection of a DNN model.

In traditional software, one potential reason contributes to the effectiveness of coverage-guided fuzzing in increasing coverage lies in that the coverage criteria follow the source code and logic structure. For example, in Fig. 1, one new input 'bac' can cover the left two branches of the program. By mutating 'bac', it is more likely to generate new input 'bad' that covers the last branch (under the previous two branches) and triggers the bug. Hence, traditional strategy usually prefers selecting new inputs as it is more likely to cover new branches based on mutating these inputs. Following the similar idea, existing DNN testing strategies (*e.g.*, *DeepTest*, *TensorFuzz*) tend to prefer selecting new inputs during the CGF process. However, it remains uncertain whether this strategy is still effective for DNN testing.

Another challenge is that there is no explicit oracle for the newly generated tests during DNN testing. The current best practice generates tests through transformations with the intention that a new test and its correspondence before the transformation should share the same semantics from a human perspective. For example, adding a little perturbation to an image of a cat would not change the human decision, so if a DNN is unable to correctly recognize the cat image after transformation, it reveals a defect in the DNN (*e.g.*, the second cat in Fig. 1). However, the assumption here is that the newly generated tests could still be recognized by a human, which, unfortunately, is not always the case. For example, with multiple transformations, an image could as well become unrecognizable (*e.g.*, too bright or too dark) for a human, leading to an invalid input (Fig. 1). Therefore, it is important to generate valid tests under the dome of DNN software testing.

To facilitate further in-depth investigation towards addressing these challenges, a general testing framework that integrates different testing criteria and testing strategies is in dire need. In this paper, we propose *DeepHunter*, a general-purpose coverage-guided fuzz testing framework for DNNs. Specifically, we propose a metamorphic mutation strategy for generating images, which to a larger extent preserves the test input semantics before and after the mutation, and a frequency-aware seed selection strategy based on the number of times a seed has been fuzzed. *DeepHunter* is designed to be extensible in the sense that new testing criteria, seed selection strategies and mutation strategies, could be easily implemented and plugged into the framework. At this moment, *DeepHunter* integrates 5 recently proposed test criteria and 4 seed selection strategies. We leverage *DeepHunter* to answer the following research questions:

- **RQ1 (Metamorphic Mutation):** How effective are different mutation strategies for generating images that keep the same semantics with the original image from a human's perspective?
- **RQ2 (Coverage):** As DNN-based models are fundamentally different from traditional software, compared with random testing, is the CGF still effective for improving coverage in DNN testing? How effective are the different seed selection strategies for improving coverage under different criteria?
- **RQ3 (Error Detection):** How different are the existing criteria for guiding erroneous behaviors detection? How effective are different seed selection strategies for detecting erroneous behaviors of DNNs? How diverse are the erroneous behaviors detected by different strategies?
- **RQ4 (Platform Migration):** Is *DeepHunter* applicable to detect the specific defects introduced by DNN quantization during platform migration?

To answer these questions and evaluate the effectiveness of *DeepHunter*, we have performed a large-scale study on 5 DNN models with diverse complexities. First, the user study on mutation strategies shows that it is difficult to always preserve the validity of the images during transformation. Even though, with careful transformation constraint design and tuning, it can still reduce the likelihood to generate invalid tests to some extent (*e.g.*, *DeepHunter* generates 1.2% on MNIST, 2.0% on CIFAR-10, and 0.8% on ImageNet). Second, although the testing criteria have no explicit structure, our results demonstrate that the CGF is still more effective than random testing in terms of coverage improving and defect detection, especially for those criteria difficult to cover. Third, compared with different seed selection strategies, we have found that selecting seeds diversely is more effective than selecting the new seeds for improving coverage and detecting defects. For example, for KMNC in LeNet-5, *DeepHunter* has achieved 69.14% coverage and detected 1,207 errors. On the contrary, *DeepTest* has only achieved 29.58% coverage and detected 208 errors, and *TensorFuzz* has achieved 55.33% coverage but only detected 618 errors. Furthermore, we have applied *DeepHunter* to detect defects introduced by DNN quantization during platform migration. The results demonstrate the usefulness of *DeepHunter* in detecting such defects even when

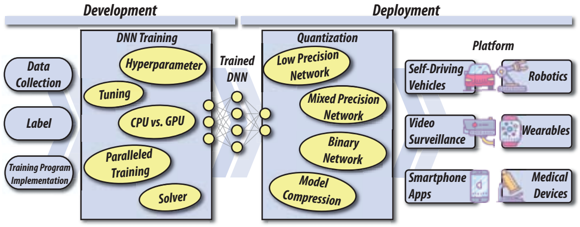

Figure 2: Two key phases in DL software life cycle.

only 10% of weights in the DNN are truncated from 32-bit to 16-bit floating-point. For example, with TKNC [31], *DeepHunter* detected 394 defects in MobileNet. To summarize, this paper makes the following contributions:

(1) We propose a metamorphic mutation strategy towards generating valid test cases. We demonstrate its effectiveness by a user study on three popular datasets.
(2) We propose a general-purpose CGF framework *DeepHunter* for testing DNNs. 5 state-of-the-art testing criteria, 3 existing and 1 new seed selection strategies are implemented in *DeepHunter*.
(3) We perform a systematically large-scale study to evaluate the effectiveness of different testing strategies and criteria on increasing coverage and detecting defects of DNNs. The results also demonstrate that *DeepHunter* is more effective than the state-of-the-art tools.
(4) We further perform a case study to demonstrate the usefulness of *DeepHunter* in detecting minor defects introduced by quantization during DNN platform migration.

2 PRELIMINARIES

DNN Training. Software development methodology [38, 40] has been well-established for traditional software, with accumulated experience and practices over the past several decades. However, different from traditional software, deep learning defines a new data-driven programming paradigm, and keeps its artifact in the form of an encoded DNN model. Such unique features bring new challenges for quality assurance of DNN based software. In particular, the decision logic of traditional software is directly written by a developer in source code. On the contrary, the major effort of a DL developer is to collect representative data and labels, design the DNN architecture, and implement the training program that specifies the runtime training behaviors. The DNN decision logic is then automatically shaped through the model training (see Fig. 2).

DNN Quantization. Once an applicable DNN model is built, it will oftentimes go through a customization phase to cater to specific software and hardware constraints of an end-user platform, such as self-driving cars and smartphones. Quantization reduces the precision of a DL model so as to improve the computation efficiency, reduce memory consumption, and storage size (see Fig. 2). Model quantization has been widely studied and becomes a common practice when migrating a large model trained on the cloud system to a mobile or IoT devices, in order to meet the extreme memory and computation requirements. Previous studies [13–15, 19, 21–24, 48, 52] have shown that quantizing the weights to lower bits (*e.g.*, from 32-bit floating to 16-bit, 8-bit) greatly reduces the model size and energy consumption, while maintaining a similar level of accuracy.

DNN Testing Criteria. Testing criteria are widely adopted in traditional software to evaluate the test quality, and to guide the test generation for defect detection. Recently, several testing criteria are proposed specialized for DNNs [31, 36]. The design of these criteria takes into account the DNN structure, which monitors the neuron activities and intrinsic network connectivity at various granularity levels. However, it is still unclear if covering the targets of these criteria indeed helps defect detection during the development and deployment of the DNNs. Furthermore, it is not obvious whether generating tests guided by these criteria exceeds the one without coverage guidance. To answer these questions, we perform a large-scale controlled study on the following testing criteria [31, 36].

- **Neuron Coverage (NC)**. NC bisects a neuron's state into *activated* and *non-activated*. Given an input, a neuron is activated if its output value is above a predefined threshold. NC measures the ratio of activated neurons of a DNN.
- ***k*-Multisection Neuron Coverage (KMNC)**. For each neuron, the range of its values (obtained from training data) are partitioned into *k* sections. An input covers a section of a neuron if the output value falls into the corresponding value section range. KMNC measures the ratio of all covered sections of all neurons of a DNN.
- **Neuron Boundary Coverage (NBC)**. Similar to KMNC, NBC analyzes the value range of a neuron covered by training data, and measures to what extent the corner-case regions outside major functional range of a neuron [31] are covered.
- **Strong Neuron Activation Coverage (SNAC).** Similar to NBC, for each neuron, SNAC considers the value range that is above the maximum value seen during training. SNAC measures how the upper corner-case regions of neurons are covered.
- **Top-*k* Neuron Coverage (TKNC)**. TKNC is a layer level testing criterion, which measures the ratio of neurons that have once been the most active *k* neurons of each layer on a given test set.

3 METHODOLOGY

In traditional software, a typical CGF iteratively performs the following steps [17]: (1) select seeds from the seed queue; (2) mutate the seed a certain number of times to generate new tests; (3) run the target program against the newly generated tests, reporting failed tests if crashes are detected, and saving those "interesting" tests, which cover new traces, into the seed queue.

Despite the large gap between traditional programs and DNNs, the success of CGF on traditional programs may still offer insights into the testing of DNNs. Intuitively, the target traditional program mirrors the target DNN, the test case of traditional program mirrors the input of the DNN, and the testing criteria [31, 36] provide the coverage metrics for testing the DNN. Considering the unique characteristics of the DNN, this paper aims to design an effective CGF framework towards providing quality assurance during the DNN development and deployment process.

3.1 Overview of DeepHunter

In general, *DeepHunter* takes a set of initial seeds and a DNN as the input. It maintains a seed queue and generates: 1) passed tests that maximize the coverage and 2) failed tests which are incorrectly predicted by the DNN. Fig. 3 depicts the overview of *DeepHunter*,

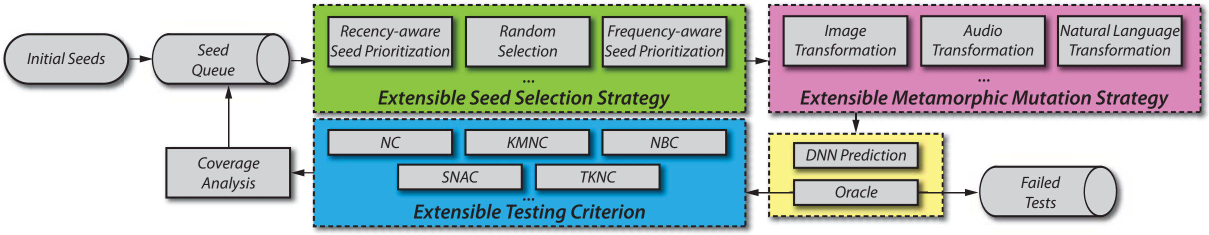
Initial Seeds
Seed Queue
Recency-aware Seed Prioritization
Random Selection
Frequency-aware Seed Prioritization
...
Extensible Seed Selection Strategy
Image Transformation
Audio Transformation
Natural Language Transformation
...
Extensible Metamorphic Mutation Strategy
Coverage Analysis
NC
KMNC
NBC
SNAC
TKNC
...
Extensible Testing Criterion
DNN Prediction
Oracle
Failed Tests

Figure 3: The workflow of *DeepHunter* with extensible seed selection strategy, mutation strategy and coverage guidance.

```
Algorithm 1: DeepHunter test generation
    input  : I: Initial Seeds, DNN: Target Neural Network
    output : F: Failed Tests
    const  : K: A configurable total number of seed mutation
 1  $F \leftarrow \emptyset$;
 2  $Q \leftarrow I$;
 3  while $s \leftarrow$ SelectNext($Q$) do
 4      $T \leftarrow$ MetamorphicMutate($s$, $K$);
 5      $cov, result \leftarrow$ Run($DNN$, $T$) ;
 6      for $s' \in T$ do
 7          if isFailedTest($s'$, $result$) then
 8              $F \leftarrow F \bigcup \{s'\}$;
 9          else if isCoverageGain($cov$) then
10              $Q \leftarrow Q.append(s')$;
```

and Alg. 1 specifies the main algorithm. At a high level, *DeepHunter* consists of 3 major components: *Extensible Seed Selection Strategy*, *Extensible Metamorphic Mutation*, and *Extensible Testing Criterion*.

During each fuzzing process, the fuzzer first selects one seed from the seed queue (Line 3). Seed selection strategy can significantly affect the effectiveness of the fuzzer [8]. We implement two seed selection strategies (*c.f.* Section 3.4) in *DeepHunter*, *i.e.*, selecting the seed randomly and selecting seeds based on their fuzz frequencies. For each selected seed, the fuzzer mutates it K times and generate a set of new tests (Line 4). We propose a metamorphic mutation strategy (*c.f.* Section 3.2) to generate mutants that maintain as many semantics from the original seed as possible. *DeepHunter* provides a set of extensible testing criteria for coverage analysis (*c.f.* Section 3.3). For example, the neuron coverage [36], k-multisection coverage [31], *etc.* After predicting the new tests with the DNN, *DeepHunter* analyzes the corresponding coverage (Line 5). For each test s', if it is a failed test, it is added into the failed set (Line 8). If a passed test increases the coverage, it is added into the end of the queue (Line 10). Otherwise, the seed is discarded.

3.2 Transformation and Mutation

3.2.1 Test Oracle. Traditional fuzzers (*e.g.*, AFL [1]) mutate a seed with bitwise/bytewise flips, block replacement, crossover between input files, *etc.* With arbitrary mutations, diverse tests are generated to maximize the coverage. The mutants are failed tests if they cause abnormal behaviors (*e.g.*, crash). However, the mutators are not useful in DNN testing as there are no such explicit oracles to judge the results of the mutants. For example, the mutators may generate images that are even unrecognizable for human (*e.g.* the invalid input in Fig. 1). This work adopts the metamorphic relation of test image during transformation as the oracle [10].

Definition 1. Given an ideal human oracle O and a test x of a specific input domain, we define *metamorphic mutation* M on x, if $\forall x' \in M(x)$, we have $O(x') = O(x)$.

Metamorphic mutation creates an oracle, *i.e.*, the semantics of the mutant x' is the same with x from human perspective. For DNN testing, the erroneous behavior of a model and the quantization erroneous behavior between two models can be checked as follows:

Definition 2. Given a DNN F, a human oracle O, a metamorphic mutation strategy M and a test x satisfying $F(x) = O(x)$, a mutant $x' \in M(x)$ is an *erroneous behavior* of F if $F(x') \neq F(x)$.

Definition 3. Given a DNN F and its quantized version F', a human oracle O, a metamorphic mutation strategy M and a test x satisfying $F(x) = O(x) \land F'(x) = O(x)$, a mutant $x' \in M(x)$ is a *quantization erroneous behavior* between F and F' if $(F(x) = F(x') \lor F'(x) = F'(x')) \land F(x') \neq F(x)$.

Intuitively, Def. 2 judges whether a new test triggers an *erroneous behavior* of DNNs if its prediction result is incorrect. Def. 3 detects the *quantization erroneous behavior* if the prediction of a test is correct in one model but incorrect in the other. For simplicity, the rest of the paper uses *error* to denote *erroneous behavior*.

Actually, a perfect metamorphic mutation strategy that guarantees to generate semantics-preserved tests under all possible cases may not exist. In this paper, we apply the domain-specific knowledge to design a conservative mutation strategy that allows the generation of semantics-preserved tests with low false positives. Since the mutators of images, voices and natural language are domain-specific and quite different, this paper focuses on the image domain, one of the most widely studied domains. The idea can be generalized to other domains.

To design an effective mutation strategy for the CGF, one challenge is to maintain the balance between *increasing the changeability of mutation* and *generating semantics-preserved tests.* If the mutation strategy changes the seed by a very small amount, the newly generated tests may be almost unchanged; Thus the fuzzer has lower chances of finding failed tests or improving coverage. If it changes the seed too much, the generated tests are likely to change the semantics. For this challenge, we proposed a new metamorphic mutation strategy for the image processing domain.

3.2.2 Metamorphic Mutation. To *increase the changeability of mutation*, we select eight image transformations of two categories:

- *Pixel Value transformation* $\mathcal{P}$: image contrast, image brightness, image blur, and image noise.
- *Affine transformation* $\mathcal{G}$: image translation, image scaling, image shearing, and image rotation.

Intuitively, *Pixel Value transformation* changes the pixel values while *Affine transformation* (*AF*) moves the pixels of the image. The transformations have been proven to be effective and useful in [44].

Algorithm 2: Metamorphic mutation

```
   input  : $s$: Seed, $K$: Total number of tests to be generated
   output : $T$: A set of new generated tests
   const  : $TRY\_NUM$: The maximum number of trials
1  $(s_0, s_0') \leftarrow info(s)$;
2  $T \leftarrow \emptyset$;
3  for $j$ from 1 to $K$ do
4      $Success \leftarrow False$;
5      for $i$ from 1 to $TRY\_NUM$ do
6          if $s_0'$ is the same with $s_0$ then
7              $t \leftarrow randomPick(\mathcal{G} \bigcup \mathcal{P})$;
8          else
9              $t \leftarrow randomPick(\mathcal{P})$;
10         $p \leftarrow pickRandomParam(t)$;
11         $s' \leftarrow t(s, p)$;
12         if $isSatisfied\big(f(s_0, s')\big)$ then
13             if $t \in \mathcal{G}$ then
14                 $s_0' \leftarrow t(s_0, p)$;
15                 $info(s') \leftarrow (s_0, s_0')$;
16             $Success \leftarrow True$;
17             $T \leftarrow T \bigcup \{s'\}$;
18             break;
19     if not $Success$ then
20         $T \leftarrow T \bigcup \{s\}$;
21 return $T$;
```

Definition 4. An image s' is *one-time mutated* from s if s' is generated after a transformation t on s (denoted as $s \xrightarrow{t} s'$), where $t \in \mathcal{P} \bigcup \mathcal{G}$. An image s' is *sequentially mutated* from s if s' is generated after a sequence of one-time mutations ($s \xrightarrow{t_0} s_1, s_1 \xrightarrow{t_1} s_2, \dots, s_n \xrightarrow{t_n} s'$) (denoted as $s \xrightarrow{t_0, t_1, \dots, t_n} s'$).

By setting conservative parameters for different transformations, it is assumed that the new image keeps the same semantics with the original image after a *one-time* mutation. However, during fuzzing, one image can be sequentially mutated from the initial seed, making it difficult to keep its validity after a sequence of mutations.

In order to *keep the semantics of the mutants* close to the original seed, we adopt a conservative strategy that selects the *Affine Transformation* to be used only once (we assume that mutation with only one affine transformation will not affect the semantics with the carefully selected parameters) as an image is more likely to be unrecognizable if it is changed by multiple affine transformations. A *Pixel Value Transformation* can be used multiple times for increasing the changeability, we use L_0 and L_∞ to constrain the pixel-level changes. Suppose an image s' is mutated from s by a pixel value transformation, then s' is valid if $f(s, s')$ is satisfied.

$$f(s, s') = \begin{cases} L_\infty(s, s') \leq 255, & \text{if } L_0(s, s') < \alpha \times \text{size}(s) \\ L_\infty(s, s') < \beta \times 255, & \text{otherwise} \end{cases} \quad (1)$$

where $0 < \alpha, \beta < 1$, $L_0(s, s')$ represents the maximum number of the changed pixels between s and s', L_∞ represents the maximum value of a pixel changes, size(s) is the total number of pixels in s.

Intuitively, if the number of changed pixels is very small (less than $\alpha \times \text{size}(s)$), we assume it does not change the semantics and L_∞ can be any value. If the number of changed pixels exceeds the boundary, we limit the maximum value (less than $\beta \times 255$) that a pixel can change. For the new image that is mutated by the *AF*, its pixels may lose the one-to-one corresponding linkage with the original image. To compute its L_0 and L_∞, we define:

Definition 5. Given an image s, which is one-time mutated or sequentially mutated from s_0 (an image in initial seeds), *i.e.*, $s_0 \xrightarrow{t_0, \dots, t_n} s$, where $n \geq 0$, the *reference image* (denoted as s_0') is defined as:

$$s_0' = \begin{cases} s_j, & \exists\, 0 \leq j \leq n.\ t_{j-1} \in \mathcal{G} \wedge s_0 \xrightarrow{t_0, \dots, t_{j-1}} s_j \\ s_0, & \text{otherwise} \end{cases}$$

The *reference image* is either the original image (if there is no *Affine Transformation*) or the generated image after an *Affine Transformation*. Note that there is only one *Affine Transformation* during sequential mutations. For sequential mutations that include an *Affine Transformation* t_{j-1} such as $s_0 \xrightarrow{t_0, \dots, t_{j-1}} s_j \xrightarrow{t_j, \dots, t_{n-1}} s_n$, the L_0 and L_∞ between s_0 and s_n are computed as follows:

$$\begin{aligned} L_0(s_0, s_n) &= L_0(s_0, s_{j-1}) + L_0(s_j, s_n) \\ L_\infty(s_0, s_n) &= MAX(L_\infty(s_0, s_{j-1}), L_\infty(s_j, s_n)) \end{aligned} \quad (2)$$

3.2.3 Metamorphic Mutation Algorithm. Alg. 2 shows the details of the metamorphic mutation, which takes an image s and the total number of images to be generated K as the input, to create a set of newly generated tests T as the output. We first obtain the initial seed s_0 and the *reference image* s_0' (Line 1). Then K new tests are generated (Line 3- 20). To generate a test, *DeepHunter* tries to mutate the seed s with a maximum number of trials TRY_NUM (Line 5-18). If the *reference image* is the same with the seed, it means the *Affine Transformation* was not adopted (*c.f*. Definition 5). In this case, both *Affine Transformation* and *Pixel Value Transformation* can be selected (Line 7). Otherwise, it can only use a pixel value transformation (Line 9). For the selected transformation t, it picks a parameter randomly (Line 10) and performs the transformation (Line 11). *DeepHunter* computes L_0 and L_∞ between the reference image s_0' and the new mutant s' to check whether s' is meaningful (Line 12). If the *Affine Transformation* is selected, it updates the *reference image* (Line 14- 15). If there is no successful mutation after TRY_NUM trials, it adds the original image s into T (Line 20).

3.3 Extensible Testing Criteria

A dumb fuzzer without any coverage guidance aimlessly mutates the seed, without knowing whether the generated test input is preferable. Consequently, such a fuzzer may frequently keep seeds that do not bring new desired information; even worse, mutation on these seeds may bury other "interesting" seeds in the fuzzing queue, largely decreasing the fuzzing effectiveness. Therefore, modern fuzzers for traditional software often embrace some feedback such as *code coverage*.

Currently, there are a variety of criteria proposed for measureing the inner behaviours of neural networks. In this paper, we implement 5 existing criteria [31, 36] as different feedback metrics to determine whether the newly generated tests should be kept for further mutation. The criteria represent the state-of-the-arts and have been proven to be useful to capture the internal DNN states. The detailed descriptions about the criteria can be found in Section 2.

3.4 Seed Prioritization

Seed prioritization decides which seed should be picked next. In traditional programs, if a seed covers a branch, it is more likely to cover the following branches by mutating the seed because there is

Table 1: Subject datasets and DNN models.

Dataset	DNN Model	#Neuron	#Layer	Acc.(%)
MNIST	LeNet-1	52	7	97.6
	LeNet-5	268	9	99.0
CIFAR-10	ResNet-20	2,570	70	91.7
	VGG-16	12,426	17	92.8
ImageNet	MobileNet	38,904	87	87.1

a hierarchical relationship between branches (*e.g.*, Fig. 1). Hence, traditional coverage-guided fuzzers usually prefer to select the new generated tests from the queue. The current techniques [35, 44] follow the similar idea to prioritize selection of the new generated tests. However, the effectiveness of such a strategy in DNN testing is still not thoroughly investigated.

Besides these existing strategies, we also implement two strategies in *DeepHunter*: 1) a uniform sampling strategy that randomly selects a seed from the queue. 2) a new strategy that probabilistically selects a seed based on the number of times it has been fuzzed. The probability of selecting a seed s is computed by:

$$P(s) = \begin{cases} 1 - g(s)/\gamma, & \text{if } g(s) < (1 - p_{\min}) \times \gamma \\ p_{\min}, & \text{otherwise} \end{cases} \tag{3}$$

where $g(s)$ represents how many times the seed has been fuzzed and $p_{\min} > 0$ is a minimum probability for selecting a seed. γ is a parameter that constrains the decrease ratio of the probability, and $p_{\min}$ denotes the minimum probability that a seed can be selected.

DeepHunter balances the diversity and recency of seed selection. Specifically, *DeepHunter* traverses the seed queue and determines whether the current seed is selected based on its probability. It applies a bias towards the new tests by assigning them with high probability. For example, the probability of new test is 1 since it gains new coverage and is regarded as “interesting”. To keep the diversity, other tests that have been fuzzed many times also have a minimum probability $p_{\min}$ to be selected.

4 EXPERIMENTAL DESIGN AND SETTINGS

We have implemented *DeepHunter* as a self-contained fuzz testing framework in Python based on deep learning framework Keras (ver.2.1.3) [11] with TensorFlow (ver.1.5.0) backend [4]. With *DeepHunter*, we perform a large-scale comparative study to answer the *four research questions* posted in Section 1. For **RQ1**, we have designed a user study to manually check the quality of the generated images (*i.e.*, whether the generated test images preserve validity). For **RQ2** and **RQ3**, we have conducted a large-scale controlled experiment to quantitatively compare the capability of coverage increase and error detection with different coverage guidance and seed selection strategies. For **RQ4**, we have run *DeepHunter* with different coverage guidance to detect errors during DNN quantization of platform migration.

Datasets and DNN Models. We select three popular publicly available datasets (*i.e.*, MNIST [29], CIFAR-10 [27], and ImageNet [41]) as the evaluation subject datasets (see Table 1). For each dataset, we study several popular pre-trained DNN models used in previous work [9, 28, 31, 49], which achieve competitive test accuracy.

User Study Settings. For **RQ1**, we perform the user study on 3 different strategies that constrain the generation of test images, and evaluate their effectiveness in generating valid test images:

- *DeepHunter*. Constrains the transformation using Equation 1.
- *TensorFuzz*. Constrains the transformation using L_∞.
- *DeepTest*. Generates images by different transformations with conservative configurations.

For *DeepHunter*, we set $\alpha = 0.02$ and $\beta = 0.2$ for Eq. 1. For *TensorFuzz* [35], we use its configuration of L_∞ (*i.e.*, 0.4). For *DeepTest* [44], it generates images and filters out some results based on the prediction results (*i.e.*, mean squared error). However, the filter is not useful in the classification task, restraining us to apply it.

For each dataset, we randomly select 30 images as the seeds and generate 5,000 images with each of the 3 mutation strategies. Finally, 9 sets, with a total of 45,000 test images (=3 datasets x 3 strategies x 5,000 generations) are generated. We recruit 9 participants for the user study and assign one test set for each participant. Each participant is asked to perform manual analysis on each of the generated test images in line with its original counterpart. A generated image is marked invalid if it either could not be perceived, or is recognized as a different class compared with its original counterpart.

Controlled Experiments. To answer **RQ2** and **RQ3**, we perform a large-scale controlled study on the effects of testing criteria and seed selection strategies. The controlled experiments are designed by considering the two aspects: 1) whether existing testing criteria are helpful for guiding error detection, and 2) how the seed selection strategy influences the testing effectiveness (including the coverage, the number, and the diversity of the errors detected) when equipped with different testing criteria. We have intensively evaluated the following 5 strategies, with the combination of different testing criteria as guidance.

(1) *Random testing (RT) without coverage guidance.* We use this as a baseline to compare with the coverage-guided strategies.
(2) *DeepHunter+Uniform (DH+UF).* Different from RT, this strategy guides RT using different testing criteria as feedback. A new test is put back to the seed queue if it improves the coverage.
(3) *DeepHunter+Probability (DH+Prob)*, adopts the probabilistic seed prioritization strategy in *DeepHunter* with coverage guidance.
(4) *DeepTest seed selection strategy* [44] with coverage guidance. In each iteration, the *last* seed is picked from the queue, based on which new tests are generated. A new test is added to the end of the queue if it improves the coverage. For a selected seed, if all of the generated tests cannot improve the coverage, the seed will be removed from the queue. Note that the seed queue may become empty in this strategy when the coverage is not improved with all the tests.
(5) *TensorFuzz seed selection strategy* [35] with coverage guidance. Suppose the size of the queue is $n > 0$. In each iteration, the strategy first constructs a reservoir of $m \leq n$ seeds, which contains one seed randomly selected from the whole queue and other $m - 1$ seeds picked from the rear of the queue. Then a seed is randomly selected from the reservoir for further test generation. Our evaluation follows the default configuration of m in *TensorFuzz* (*i.e.*, $m = 6$).

We eventually configure 21 different fuzzers (5 testing criteria × 4 seed selection strategies + 1 RT with no coverage guidance). To counter the randomness effect during fuzzing, we repeat the execution of each fuzzer 10 times and average the results. Due to the huge execution cost (each model is fuzzed with 21 × 10 times),

Table 2: The number and ratio of invalid images generated by different mutation strategies

Strategies	MNIST	CIFAR-10	ImageNet
DeepTest	595 (11.9%)	1600 (32.0%)	915 (18.3%)
TensorFuzz	155 (3.1%)	370 (7.4%)	255 (5.1%)
DeepHunter	60 (1.2%)	145 (2.9%)	40 (0.8%)

we select the 4 models in MNIST and CIFAR-10 in the controlled experiments For each dataset, we randomly selected 1,000 tests from their original test data as initial seeds such that these tests are correctly handled by all the models. We ran each fuzzer with the same number of iterations (*i.e.*, 5,000) and used the same metamorphic mutation set.

Experiments on Quantization Error Detection. In this experiment, we investigate whether *DeepHunter* is useful for detecting errors introduced during the quantization (**RQ4**), which is often more difficult than detecting prediction errors of a DNN model.

We select one DNN model for each of the three datasets (*i.e.*, LeNet-5, ResNet-20, and MobileNet). For each model (originally in 32-bit floating point precision), we perform quantization with 3 configurations: (1) randomly sampling 10% of weights and truncating 32-bit floating point to 16-bit, (2) randomly sampling 50% weights and truncating 32-bit floating point to 16-bit, and (3) truncating all weights from 32-bit floating point to 16-bit. To reduce the effect of the randomness during the mixed precision quantization, we randomly generate 10 different quantized models for each of the 10% and 50% sampling cases. At runtime, we allocate 10 hours for test generation on each original unquantized model. In addition, we repeat each configuration 5 times and average the results.

Other Parameter Configuration. In all experiments, we set the threshold of NC to 0.75 following *DeepXplore*. For KMNC, we set the k to 1,000, and follow the default setting for SNAC, NBC, and TKNC used in *DeepGauge*. *TRY_NUM* in Alg. 2 is set as 50. P_{min} and γ in Equation 2 are set to 0.5 and 20, respectively. The parameter K in Alg. 1 is set as 20. All the experiments are preformed on a server with the Ubuntu 16.04 system with 28-core 2.0GHz Xeon CPU, 196 GB RAM and 4 NVIDIA Tesla V100 16G GPUs.

5 EXPERIMENTAL RESULTS

5.1 RQ1. User Study on Metamorphic Mutation

Table 2 shows the number of invalid images generated by different strategies through manual analysis. The invalidity rate of CIFAR-10 is generally higher than MNIST and ImageNet among all of the three strategies. The reason is that, the resolution of CIFAR-10 is relatively low. As a result, a CIFAR-10 image is difficult to be recognized by human, even if it is a valid input that can be correctly predicted by the DNN.

Although the parameters of transformations used in *DeepTest* are rather conservative, it generates many invalid images without constraints. For example, the invalid images take up 32.0% and 18.3% for CIFAR-10 and ImageNet, respectively. With a further transformation constraint (*i.e.*, within L_∞ norm), *TensorFuzz* reduces the invalid images generation ratio for many cases. The conservative setting in *DeepHunter* confines the invalidity ratio to be even lower. For example, the invalidity ratio is reduced to 2.9% for CIFAR-10 and 0.8% for ImageNet. Note that the purpose to perform the user study on these mutation strategies is not to compare these strategies since there is no best configuration for all cases. Instead, we intend to show that the careful design and parameter tuning of the mutation strategy indeed help to reduce the invalidity image ratio for particular application scenarios. We use these parameters in the following experiments.

> **Answer to RQ1:** The user study shows that each metamorphic mutation strategy may generate invalid images. With proper constraint design and parameter tuning, it is possible to further reduce the invalidity ratio. Our mutation strategy generates test images with a higher validity ratio.

5.2 RQ2. Results of Coverage Increase

Table 3 shows the average results of coverage under different criteria. The row Init. shows the coverage achieved by the initial seeds. The bold number denotes the best case for a specific criterion using different strategies. Overall, we have the following findings:

●*Despite fundamentally different from traditional software, CGF is still effective to maximize coverage than random testing, especially for those criteria that are difficult to cover.* Comparing the results between coverage-guided testing strategies and random testing without coverage guidance (i.e., RT), in most cases, CGF achieves higher coverage than RT (especially for NBC and SNAC). For example, for LeNet-5, the coverage achieved by RT is only 1.41%, 0.92% for NBC and SNAC; With coverage-guidance, *DH+UF* achieves 6.23% (4x+) and 10.56% (9x+) while *DH+Prob* achieves even higher results. For other criteria that are relatively easier to cover, RT obtains competitive results with CGF. Although existing DL testing criteria do not exhibit explicit structural information like those in source code, these criteria might still incorporate some hidden decision flow information that makes coverage-guided strategies effective in improving coverage.

●*Different from fuzzers of traditional software that usually prioritize selecting newly generated tests, sampling diverse seeds from the queue is also important in DNN testing.* Overall, *DH+UF* and *DH+Prob* obtain higher coverage than *Tensorfuzz* and *DeepTest* in many cases. The latter two prefer selecting new generated tests while the former two tend to select more diverse seeds during fuzzing. As an example, Fig. 4 plots the coverage increasing trends of LeNet-5 under criteria KMNC and NBC. The complete plots of all configurations are put on the website [2]. Although *DeepTest* terminates earlier than the given 5000 iterations on the difficult criteria NBC, it still achieves better coverage in its first 1000 iterations, indicating that selecting new test might be helpful for improving difficult criteria. *DH+Prob* balances between randomly selecting seeds and selecting new generated tests. As a result, *DH+Prob* achieves the highest coverage in many cases. In addition, an interesting results on *DeepTest*'s runtime behavior draw our attention. In particular, *DeepTest* runs 5000 iterations for KMNC but achieves the worst result for all models. Our in-depth analysis reveals that: KMNC measures the major behaviors of DNNs that are more likely to be captured by diverse inputs. However, for KMNC guidance on LeNet-5, *DeepTest* often selects the last one from the seed queue, based on which some of the new mutants still increase KMNC coverage. Thus, the new tests are further put into the rear of the queue and will be selected in

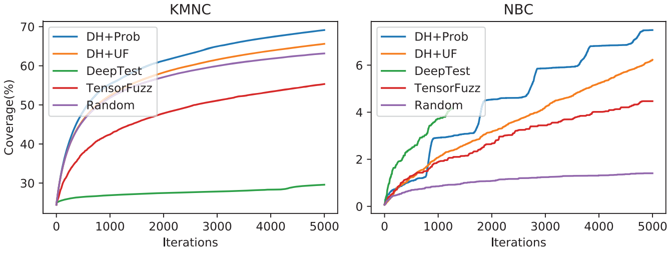

Figure 4: Coverage increasing on LeNet-5 by fuzzers with different configurations averaged over 10 runs

Table 3: Average results (%) of coverage over 10 runs by fuzzers with different configurations

Model	Strategies	KMNC	NBC	SNAC	TKNC	NC
LeNet-1	Init.	35.19	0.00	0.00	88.57	23.81
	DH+Prob	**87.84**	16.28	**15.85**	**88.57**	**26.19**
	DH+UF	87.57	**16.38**	11.30	**88.57**	**26.19**
	DeepTest	45.90	13.34*	5.97*	**88.57***	25.24*
	TensorFuzz	82.12	13.81	10.56	**88.57**	24.29
	Random	75.02	7.68	3.03	**88.57**	23.81
LeNet-5	Init.	24.46	0.07	0.14	75.47	58.53
	DH+Prob	**69.14**	**7.49**	**10.56**	85.11	**70.31**
	DH+UF	65.61	6.23	8.93	**85.24**	70.16
	DeepTest	29.58	4.16*	4.23*	83.75*	68.53*
	TensorFuzz	55.33	4.47	5.14	83.67	67.83
	Random	63.15	1.41	0.92	82.30	67.36
ResNet-20	Init.	39.81	0.21	0.24	56.91	10.72
	DH+Prob	**74.70**	**13.92**	**19.23**	**68.78**	20.00
	DH+UF	70.41	12.62	16.09	68.09	**20.30**
	DeepTest	41.85	13.05	18.27	67.91*	16.92*
	TensorFuzz	62.33	8.82	11.93	66.16	20.22
	Random	69.87	2.98	3.42	64.23	16.15
VGG16	Init.	46.74	0.19	0.21	10.21	40.91
	DH+Prob	**85.74**	10.01	**18.61**	**16.95**	50.45
	DH+UF	83.15	**11.7**	17.37	16.6	48.29
	DeepTest	55.22	3.57	3.71	13.34	48.87*
	TensorFuzz	78.19	9.58	13.66	16.03	**51.22**
	Random	81.08	3.34	3.56	15.08	46.37

∗ means *DeepTest* terminates early than the given 5000 iterations because the seed queue becomes empty if none of the generated tests could improve coverage.

the next iteration. As a result, the strategy reduces the diversity of seed selection and the coverage increases slowly.

- *The difficulty in enhancing coverage of different criteria is different.* For KMNC, TKNC, and NC, it is relatively easier to improve coverage. The obtained coverage of these criteria is higher than the rest. Actually, the initial seeds already achieve a certain level of coverage. For example, in LeNet-1, initial seeds achieve very high coverage (88.57%) in TKNC, the later testing process does not increase it. For easier test criteria, different strategies achieve similar coverage results. For NBC and SNAC, it is more difficult to cover since they represent the corner-regions. Such results are consistent with the coverage trends reported in [31], where the initial seeds obtain rather low coverage.

Answer to RQ2: Coverage-guided fuzz testing is more effective in improving coverage than random testing, especially for criteria that are difficult to cover. *DH+Prob*, that balances both new and diverse seed selection diversity, is oftentimes more effective than other strategies across different criteria.

Table 4: Average number of unique errors detected over 10 runs by fuzzers with different configurations

Model	Strategies	KMNC	NBC	SNAC	TKNC	NC
LeNet-1	DH+Prob	**1530.3**	**2287.1**	**2375.0**	**2354.4**	**2350.4**
	DH+UF	1503.8	2157.0	2201.0	2229.0	2233.0
	DeepTest	149.6	693.3	638.0	632.4	634.3
	TensorFuzz	842.7	626.2	1464.9	626.2	627.4
	Random			439.0		
LeNet-5	DH+Prob	**1207.0**	2403.3	2529.4	**2627.0**	**2833.3**
	DH+UF	664.2	**2510.0**	**2589.2**	2570.5	2730.4
	DeepTest	208.1	706.6	692.8	789.0	692.0
	TensorFuzz	618.5	618.8	590.0	981.0	1635.2
	Random			482.3		
ResNet-20	DH+Prob	**4070.5**	4372.9	4726.0	**5213.0**	**6228.5**
	DH+UF	2900.3	**5034.2**	**5302.2**	5100.8	5957.5
	DeepTest	317.2	2907.0	3194.3	2927.8	1594.7
	TensorFuzz	2121.7	3851.1	3201.7	4228.3	5808.0
	Random			2554.4		
VGG16	DH+Prob	**5139.6**	**7502.0**	**6752.2**	**10553.1**	**9378.2**
	DH+UF	3130.9	5456.7	5437.8	6426.3	7289.4
	DeepTest	471.2	3090.6	5913.4	8935.1	3657.0
	TensorFuzz	4284.1	6538.0	5436.0	6866.6	7823.9
	Random			2863.0		

5.3 RQ3. Results of Error Detection

5.3.1 Quantity. Table 4 shows the unique errors detected by different fuzzers. Random testing has no coverage guidance, resulting in only one number in the row. The bold numbers show the best coverage obtained under the same criteria guidance across different seed selection strategies.

The results are consistent with the coverage results of **RQ2**: Overall, CGF is effective in detecting more unique errors than random testing without coverage guidance. Seed selection diversity from the queue is important. For example, *DH+Prob* detects more errors than the other strategies for most cases. In smaller models, the number of errors detected by *DH+Prob* and *DH+UF* are similar. However, when model becomes more complex, *DH+Prob* detects more errors than *DH+UF*. For example, in VGG16, *DH+Prob* detects 10,553 unique errors while *DH+UF* only uncovers 5,437 unique errors.

We also observe that KMNC is less effective than the other criteria in detecting errors, likely because it mainly considers the major functional behavior coverage [31]. In KMNC, *DH+Prob* exhibits its advantage over the other strategies. For example, compared with the second best result in KMNC, *DH+Prob* achieves 81.8%, 40.3% and 16.6% improvement for LeNet-5, ResNet-20 and VGG16. *DeepTest* detects the least number of errors as it always selects the last seed of the queue. For other 4 criteria, the number of errors detected is similar and no single strategy exceeds others in all cases.

In addition, we also analyzed the number of seeds that are kept in the final seed queue. Note that the seeds in the seed queue are benign data which are predicted correctly. The results show that, for those criteria that are easy to cover, more seeds tend to be kept in the queue. For example, in LeNet-1, *DH+Prob* keeps 14119 seeds for KMNC but only keeps 1001 seeds for NC. For large models, fuzzers tend to keep more seeds because it is easier to improve the coverage of large models. For example, for KMNC, *DH+Prob* keeps 14119 seeds in LeNet-1 and 34008 seeds in LeNet-5. Due to the space limit, the complete results can be found on our website [2].

Table 5: Average number of error categories found by fuzzers with different configurations over 10 runs

Model	Strategies	KMNC	NBC	SNAC	TKNC	NC
LeNet1	DH+Prob	**731.0**	**973.0**	**976.4**	**972.8**	**977.6**
	DH+UF	380.6	964.8	970.0	972.0	974.8
	DeepTest	1.0	806.2	801.2	802.6	806.4
	TensorFuzz	81.2	459.8	467.8	488.4	489.8
	Random	179.0				
LeNet-5	DH+Prob	**539.4**	**843.4**	**871.4**	**866.6**	**895.4**
	DH+UF	257.0	831.8	839.2	825.0	873.4
	DeepTest	1.0	451.8	441.6	497.0	485.0
	TensorFuzz	67.8	302.2	306.6	289.4	312.6
	Random	206.0				
ResNet-20	DH+Prob	**724.6**	**802.0**	**855.2**	**909.0**	**952.4**
	DH+UF	320.4	786.8	840.8	848.8	927.4
	DeepTest	1.0	261.4	646.2	728.2	709.8
	TensorFuzz	81.0	277.2	296.8	338.6	432.4
	Random	288.4				
VGG16	DH+Prob	**742.0**	**753.0**	**724.0**	**743.0**	**962.0**
	DH+UF	246.0	620.0	653.0	607.0	923.0
	DeepTest	1.0	1.0	3.0	3.0	877.0
	TensorFuzz	46.0	136.0	144.0	158.0	366.0
	Random	204.0				

5.3.2 *Diversity.*

Besides the quantity, diversity of errors is important as well. Diverse errors provide more feedback that informs developers to understand the problem towards further enhancing the robustness [33]. We estimate the diversity in a way that errors are sequentially mutated from the same seed belonging to the same category. Table 5 shows the average number of error categories detected by different fuzzers. The total number of categories equals to the number of initial seeds (*i.e.*, 1000). The results show that *DH+Prob* and *DH+UF* detect more diverse errors than the other strategies. As the size of the model grows, *DH+Prob* exhibits even higher performance margin than *DH+UF* in many cases; *DeepTest* and *TensorFuzz* detect fewer error categories even than RT. For example, in VGG16, *DH+Prob* detects more than an average of 28.7% error categories than *DH+UF* among the five criteria. *DeepTest* only detects 1 category in KMNC and NBC, 3 categories in SNAC and TKNC (note that, in VGG16, *DeepTest* consumes all the 5000 iterations on these criteria). Similarly, TensorFuzz also detects fewer error categories than random testing. The reason is that both of them favor in selecting the new tests in the queue, without balancing the seed selection diversity. Consequently, when the coverage is continuously increased by such tests, the seed selection strategy tends to mutate seeds from the same category. Fig. 5 plots the average ratios of diverse errors found in VGG16 across different fuzzers under KMNC and NBC, with more complete results available on the website [2] The overall results confirms that *DH+Prob* detects errors with higher diversity efficiently.

The study reveals that the seed selection strategy is important in DNN testing. Intensively favoring in selecting new seed may reach corner cases, but result in generating tests lacking diversity. A simple random selection increases the diversity but may not find corner cases.

> **Answer to RQ3:** Coverage-guided fuzzing is effective in detecting errors of DNNs. Such effectiveness is highly influenced by the testing strategy. A balanced strategy *DeepHunter*+Prob considering both diversity and priority in seed selection often detects diverse errors efficiently.

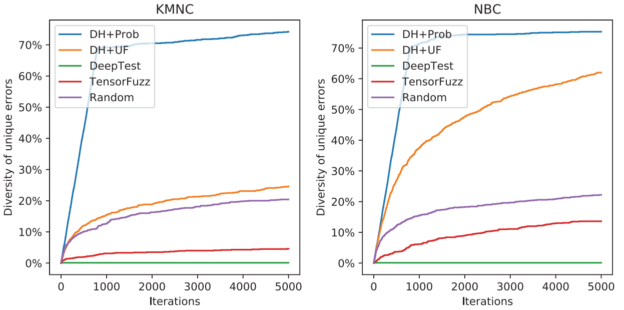

Figure 5: Average results of diverse errors found in VGG16 by different fuzzers

Table 6: Average number of unique errors detected by *DeepHunter+Prob* and *TensorFuzz* over 5 runs under different DNN quntization ratio settings from 32-bit to 16-bit.

DNN	Q.R.(%)	*DeepHunter*+Prob					TensorFuzz
		NC	KMNC	NBC	SNAC	TKNC	
LeNet-5	10	0	6.2	**10.4**	5.0	9.3	7.1
	50	3.0	8.1	11.0	3.2	8.6	**15.0**
	100	3.0	19.2	**33.2**	13.5	21.4	20.4
ResNet-20	10	11.2	19.2	**45.3**	17.6	14.2	35.1
	50	28.3	49.1	**72.1**	50.0	44.2	54.4
	100	52.2	88.4	120.3	120.0	89.9	**130.3**
MobileNet	10	274.2	190.3	307.3	319.1	**394.7**	80.6
	50	590.3	385.7	686.9	594.4	**689.0**	160.7
	100	1007.6	806.0	1175.7	1177.1	**1235.3**	360.4

5.4 RQ4. Error Detection for Quantization

Table 6 summarizes the results in detecting errors introduced during quantization by *DeepHunter+Prob* and *TensorFuzz*. In our experiment, all initial seeds of each dataset are unable to detect the errors before and after quantization for all cases.

We can see that the number of errors found by *DeepHunter* with different coverage guidance is different. With NBC, *DeepHunter* found more errors for LeNet-5 and ResNet-20. For MobileNet, errors found with NBC, SNAC, TKNC outnumbers the other two criteria. The result reveals that KMNC and NC, which represent the overall major behavior of DNNs, are relatively not effective in such scenarios. Corner-region based criteria may be more useful to capture quantization errors.

We also find that the same quantization ratio (QR) may have different effects on models with different complexity. The same QR would reduce the precision of more weights in a larger model, potentially introducing more changes on the decision logic. For example, both *DeepHunter* and *TensorFuzz* found more errors in ResNet-20 and MobileNet but less in LeNet-5. Moreover, a larger QR would often introduce larger logic decision inconsistencies for DNN models before and after quantization. This is confirmed by our results (see Table 6). Given the same model, we find the number of detected errors tends to increase by both *DeepHunter* and *TensorFuzz* as the QR increases.

> **Answer to RQ4:** With corner-region based criteria, *DeepHunter* tends to be more useful for detecting errors introduced during the DNN quantization. A larger QR often introduces larger decision logic inconsistencies before and after quantization. Similarly, the same QR would often bring higher influence in larger models.

5.5 Threats to Validity

The selection of the subject datasets and DNN models could be a threat to validity. In this paper, we try to counter this issue by using 3 well-studied datasets with diverse complexity and popular pre-trained DNN models that obtain competitive performance. It is worth noting that our testing framework scales to the practical-sized general purpose object classification dataset ImageNet. Another threat is the metamorphic mutation that could not guarantee the validity of generated images in all cases. We counter this issue by designing and tuning a conservative configuration, and perform a user study to demonstrate that our mutation strategy achieves a high validity ratio for generated test images (in **RQ1**). A further threat would be the randomness factor during test generation and mixed-precision model quantization. We counter this issue by repeating multiple times for each configuration, generating multiple quantization models, and averaging the results.

6 RELATED WORK

In this section, we summarize the relevant work about testing and verification of DL systems.

Testing. *DeepXplore* [36] originally proposes a white-box differential testing technique to detect behavior inconsistencies among various DNNs with neuron coverage guidance. *DiffChaser* [50] proposes a black-box differential testing technique to detect the disagreements. Different from *DeepXplore* and *DiffChaser* which focus on differential testing, *DeepHunter* is a grey-box testing technique focusing on a single DNN model. In the following work of *DeepXplore*, *DeepTest* [44] further leverages neuron coverage to guide testing for DNN-based autonomous driving, which adopts the domain-specific metamorphic transformation on camera images to detect erroneous behaviors of a DNN model. *DeepRoad* [51] proposes a GAN-based scene mutation transformation (*i.e.*, snowy and rainy) to test autonomous driving system. The GAN-based mutation transformation often generates images more closely to realistic scenes, but the obtaining of the mutator often requires non-trivial effort of pair-set data collection and training. A well-performed GAN mutation could be integrated into *DeepHunter* as an advanced mutator for some domain specific testing. *DeepConcolic* [43] is another recently proposed white-box DL testing technique based on concolic testing. Given a seed image, *DeepConcolic* encodes a target neuron activation coverage configuration as linear constraints, which is further resolved by a constraint solver. However, the constraint solving-based approach may face challenges in scaling to practical-sized dataset and DNN models.

DeepGauge [31] generalizes the concept of neuron coverage and proposes a set of coverage criteria that take the distribution of training data into consideration. *DeepGauge* is demonstrated to be more accurate over neuron coverage in capturing minor error behaviors introduced by four state-of-the-art adversarial test generation techniques. In [26], the surprise adequacy test criteria is proposed to measure the surprise of an input in terms of the training data. The evaluation results demonstrate that the surprise metric is useful to improve accuracy of DNN via retraining. *DeepCT* [30] considers the interactions of neurons and proposes a set of combinatorial testing criteria for DNNs. *DeepStellar* [16] further proposes a set of metrics for recurrent neural networks based on state modeling, which are applied on testing and adversarial example detection. *DeepMutation* [32] proposes the mutation testing technique of DNNs, by introducing a set of fault injection operators to perturbates the decision logic of a DNN. MODE [33] is recently developed for model debugging process, inspired by state differential analysis and input data selection in software debugging and regression testing. MODE can effectively and efficiently select the suitable portion of the training data and fix the model bugs, without much compromise on accuracy and training time cost.

A concurrent work, *TensorFuzz* [35], tries to debug neural networks with coverage-guided fuzzing. *DeepHunter* differs from *TensorFuzz* mainly in several aspects. *TensorFuzz* provides one mutator (*i.e.*, add noise to the inputs). The mutators of *DeepHunter* are more diverse and plausible, with careful design. *DeepHunter* employs a set of five recently proposed testing criteria for providing multi-faceted feedback to the fuzzer. In addition, *DeepHunter* integrates 4 seed selection strategies.

Different from the existing works, this paper proposes a coverage-guided fuzz testing framework that scales to practical-sized dataset and DNN models. The major aim of *DeepHunter* is to provide a common framework that facilitates further comparative studies on DNN testing. Our large-scale study marks the first step towards such a direction, with interesting findings identified.

Verification. The reliability of DNNs has been investigated by recent work with formal guarantees [18, 25, 37, 39, 45, 47]. Reluplex [25] adopts an SMT-based approach on a neural network of 300 ReLU nodes for safety verification. *DeepSafe* [18] tries to identify safe regions in the input space using Reluplex as its core. A more recent work AI^2 [45] adopts abstract interpretation with well designed abstract domains to verify DNNs.

This paper further pushes forward the quality assurance of DNNs from the automated testing perspective, by examining whether coverage-guided fuzz testing could be useful as a potential software quality assurance technique for DNN software.

7 CONCLUSION

This paper proposed a general coverage-guided fuzz testing framework for DNNs. *DeepHunter* is designed to be extensible, which currently integrates four seed selection strategies and five testing criteria. We performed a large-scale comparative study to evaluate the effects of different testing strategies on different criteria, and demonstrated the effectiveness of *DeepHunter*. Since the investigation on quality assurance of deep learning is still at an early stage, we hope that *DeepHunter* can benefit both the SE and the AI communities, and facilitate further extensive studies on constructing high quality DNN software.

ACKNOWLEDGMENTS

This research was supported (in part) by the National Research Foundation, Prime Ministers Office, Singapore under its National Cybersecurity R&D Program (Award No. NRF2018NCR-NCR005-0001), National Satellite of Excellence in Trustworthy Software System (Award No. NRF2018NCR-NSOE003-0001) administered by the National Cybersecurity R&D Directorate, and JSPS KAKENHI Grant 19H04086.

REFERENCES

[1] 2018. libFuzzer . (2018). https://llvm.org/docs/LibFuzzer.html

[2] 2019. DeepHunter. (2019). https://sites.google.com/view/deephunter

[3] Craig S. Smith. 2018. Alexa and Siri Can Hear This Hidden Command. You Can't. (2018). https://www.nytimes.com/2018/05/10/technology/alexa-siri-hidden-command-audio-attacks.html

[4] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 265–283.

[5] Amazon. 2018. Amazon Alexa. (2018). https://developer.amazon.com/zh/alexa

[6] Paul Ammann and Jeff Offutt. 2016. *Introduction to Software Testing* (2nd ed.). Cambridge University Press, New York, NY, USA.

[7] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2013. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 387–402.

[8] Marcel Böhme, Van-Thuan Pham, and Abhik Roychoudhury. 2016. Coverage-based Greybox Fuzzing As Markov Chain. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. ACM, New York, NY, USA, 1032–1043. DOI: http://dx.doi.org/10.1145/2976749.2978428

[9] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *Security and Privacy (SP), IEEE Symposium on*. 39–57.

[10] Tsong Y Chen, Shing C Cheung, and Shiu Ming Yiu. 1998. *Metamorphic testing: a new approach for generating next test cases*. Technical Report. Technical Report HKUST-CS98-01, Department of Computer Science, Hong Kong âĂę.

[11] François Chollet and others. 2015. Keras. https://github.com/fchollet/keras. (2015).

[12] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. 2012. Multi-column deep neural networks for image classification. In *CVPR*. 3642–3649.

[13] Matthieu Courbariaux and Yoshua Bengio. 2016. BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *arXiv preprint arXiv:1602.02830* (2016).

[14] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2014. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024* (2014).

[15] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. BinaryConnect: Training Deep Neural Networks with Binary Weights During Propagations. In *NIPS*. 3105–3113.

[16] Xiaoning Du, Xiaofei Xie, Yi Li, Lei Ma, Yang Liu, and Jianjun Zhao. 2019. DeepStellar: Model-Based Quantitative Analysis of Stateful Deep Learning Systems. In *The 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*.

[17] Shuitao Gan, Chao Zhang, Xiaojun Qin, Xuwen Tu, Kang Li, Zhongyu Pei, and Zuoning Chen. 2018. CollAFL: Path Sensitive Fuzzing. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 679–696.

[18] Divya Gopinath, Guy Katz, Corina S. Pasareanu, and Clark Barrett. 2017. DeepSafe: A Data-driven Approach for Checking Adversarial Robustness in Neural Networks. *CoRR* abs/1710.00486 (2017). arXiv:1710.00486 http://arxiv.org/abs/1710.00486

[19] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep learning with limited numerical precision. In *ICLR*. 1737–1746.

[20] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, and others. 2012. 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.

[21] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research* 18, 1 (2017), 6869–6898.

[22] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *CVPR*.

[23] F. Juefei-Xu, Vishnu Boddeti, and M. Savvides. 2017. Local Binary Convolutional Neural Networks. In *CVPR*. IEEE, 19–28.

[24] F. Juefei-Xu, V. N. Boddeti, and M. Savvides. 2018. Perturbative Neural Networks. In *CVPR*. IEEE, 3310–3318.

[25] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. *CoRR* abs/1702.01135 (2017). arXiv:1702.01135 http://arxiv.org/abs/1702.01135

[26] Jinhan Kim, Robert Feldt, and Shin Yoo. 2019. Guiding Deep Learning System Testing using Surprise Adequacy. In *The 41st ACM/IEEE International Conference on Software Engineering*.

[27] Nair Krizhevsky, Hinton Vinod, Christopher Geoffrey, Mike Papadakis, and Anthony Ventresque. 2014. The cifar-10 dataset. http://www.cs.toronto.edu/kriz/cifar.html. (2014).

[28] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. of the IEEE* 86, 11 (1998), 2278–2324.

[29] Yann LeCun and Corrina Cortes. 1998. The MNIST database of handwritten digits. (1998).

[30] L. Ma, F. Juefei-Xu, M. Xue, B. Li, L. Li, Y. Liu, and J. Zhao. 2019. DeepCT: Tomographic Combinatorial Testing for Deep Learning Systems. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 614–618.

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

[32] Lei Ma, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Felix Juefei-Xu, Chao Xie, Li Li, Yang Liu, Jianjun Zhao, and others. 2018. DeepMutation: Mutation Testing of Deep Learning Systems. *The 29th IEEE International Symposium on Software Reliability Engineering (ISSRE)* (2018).

[33] Shiqing Ma, Yingqi Liu, Wen-Chuan Lee, Xiangyu Zhang, and Ananth Grama. 2018. MODE: Automated Neural Network Model Debugging via State Differential Analysis and Input Selection. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018)*. ACM, New York, NY, USA, 175–186. DOI: http://dx.doi.org/10.1145/3236024.3236082

[34] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, and others. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[35] Augustus Odena and Ian Goodfellow. 2019. TensorFuzz: Debugging Neural Networks with Coverage-Guided Fuzzing. In *Proceedings of the Thirty-sixth International Conference on Machine Learning*.

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

[37] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Towards Practical Verification of Machine Learning: The Case of Computer Vision Systems. *CoRR* abs/1712.01785 (2017). arXiv:1712.01785 http://arxiv.org/abs/1712.01785

[38] Roger Pressman. 2010. *Software Engineering: A Practitioner's Approach* (7 ed.). McGraw-Hill, Inc., New York, NY, USA.

[39] Luca Pulina and Armando Tacchella. 2010. An Abstraction-Refinement Approach to Verification of Artificial Neural Networks. In *International Conference on Computer Aided Verification*. Springer, 243–257.

[40] Nayan B. Ruparelia. 2010. Software Development Lifecycle Models. *SIGSOFT Softw. Eng. Notes* 35, 3 (May 2010), 8–13. DOI: http://dx.doi.org/10.1145/1764810.1764814

[41] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *IJCV* 115, 3 (2015), 211–252.

[42] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, and others. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484.

[43] Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. 2018. Concolic Testing for Deep Neural Networks. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE 2018)*. ACM, New York, NY, USA, 109–119. DOI: http://dx.doi.org/10.1145/3238147.3238172

[44] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th International Conference on Software Engineering*. ACM, 303–314.

[45] Dana Drachsler-Cohen Petar Tsankov Swarat Chaudhuri Martin Vechev Timon Gehr, Matthew Mirman. 2018. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*.

[46] Uber Accident. 2018. After Fatal Uber Crash, a Self-Driving Start-Up Moves Forward. (2018). https://www.nytimes.com/2018/05/07/technology/uber-crash-autonomous-driveai.html

[47] Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. 2017. Feature-Guided Black-Box Safety Testing of Deep Neural Networks. *CoRR* abs/1710.07859 (2017). arXiv:1710.07859 http://arxiv.org/abs/1710.07859

[48] Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. 2018. Training and inference with integers in deep neural networks. *arXiv preprint arXiv:1802.04680* (2018).

[49] C. Xiao, B. Li, J.-Y. Zhu, W. He, M. Liu, and D. Song. 2018. Generating Adversarial Examples with Adversarial Networks. *ArXiv e-prints* (Jan. 2018). arXiv:cs.CR/1801.02610

[50] Xiaofei Xie, Lei Ma, Haijun Wang, Yuekang Li, Yang Liu, and Xiaohong Li. 2019. DiffChaser: Detecting Disagreements for Deep Neural Networks. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*.

[51] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. 2018. DeepRoad: GAN-based Metamorphic Testing and Input Validation Framework for Autonomous Driving Systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE 2018)*. ACM, New York, NY, USA, 132–142. DOI: http://dx.doi.org/10.1145/3238147.3238187

[52] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. 2017. Trained ternary quantization. *ICLR* (2017).