Boosting the Revealing of Detected Violations in Deep Learning Testing: A Diversity-Guided Method

Xiaoyuan Xie*† xxie@whu.edu.cn School of Computer Science, Wuhan University, Wuhan, China

Pengbo Yin* Ryannn@whu.edu.cn School of Computer Science, Wuhan University, Wuhan, China

Songqiang Chen*† i9schen@gmail.com School of Computer Science, Wuhan University, Wuhan, China

ABSTRACT

Due to the ability to bypass the oracle problem, Metamorphic Testing (MT) has been a popular technique to test deep learning (DL) software. However, no work has taken notice of the prioritization for Metamorphic test case Pairs (MPs), which is quite essential and beneficial to the effectiveness of MT in DL testing. When the fault-sensitive MPs apt to trigger violations and expose defects are not prioritized, the revealing of some detected violations can be greatly delayed or even missed to conceal critical defects. In this paper, we propose the first method to prioritize the MPs for DL software, so as to boost the revealing of detected violations in DL testing. Specifically, we devise a new type of metric to measure the execution diversity of DL software on MPs based on the distribution discrepancy of the neuron outputs. The fault-sensitive MPs are next prioritized based on the devised diversity metric. Comprehensive evaluation results show that the proposed prioritization method and diversity metric can effectively prioritize the fault-sensitive MPs, boost the revealing of detected violations, and even facilitate the selection and design of the effective Metamorphic Relations for the image classification DL software.

CCS CONCEPTS

• **Software and its engineering → Software verification and validation**.

KEYWORDS

deep learning testing, metamorphic testing, diversity, test case selection and prioritization

ACM Reference Format:

Xiaoyuan Xie, Pengbo Yin, and Songqiang Chen. 2022. Boosting the Revealing of Detected Violations in Deep Learning Testing: A Diversity-Guided Method. In *37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22), October 10–14, 2022, Rochester, MI, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3551349.3556919

*Equal contribution and co-first authors.
†Co-corresponding authors.

ASE '22, October 10–14, 2022, Rochester, MI, USA

ACM ISBN 978-1-4503-9475-8/22/10…$15.00
https://doi.org/10.1145/3551349.3556919

1 INTRODUCTION

Deep learning (DL) software, which is powered by the deep neural networks (DNNs), has now been omnipresent in daily human life to handle many complicated and abstract tasks, such as the image classification and recognition [9, 16, 28], text processing [10, 21], and speech analysis [14, 15]. However, such software also exhibits defects as traditional software does, some of which have caused fatal accidents in critical applications like the autonomous driving systems [36, 39] and the translation services [17, 31, 32]. Therefore, it is essential and has attracted a lot of effort to extensively test various kinds of DL software and effectively reveal their defects.

Currently, Metamorphic Testing (MT) has almost been the most popular technique for DL testing due to its great ability to alleviate the test oracle problem. It allows the unlabeled test cases being efficiently used to extensively test DL software [5]. Given one DL software to test, MT first constructs a few *Metamorphic test case Pairs* (MPs) based on several *Metamorphic Relations* (MRs). Each MP consists of an existing test case (called the *source case*) and a new test case (called the *follow-up case*) generated with a specific MR. Each MR defines the rule to generate the follow-up case from the source case and the expected relation between the outputs of the two cases. Next, MT executes the tested software over each MP and inspects the relation between the real outputs against MRs. Once two outputs do not satisfy the MRs, a violation is revealed to expose defects. So far, we have seen many studies propose various MRs and successfully reveal quite a few defects in various DL software, such as the autonomous driving systems [36, 39], the translation services [17, 31, 32], and the question answering systems [4, 37].

However, current works on MT for DL testing mainly focus on the exploration of new application fields and novel MRs [38]. No work has paid attention to the prioritization of MPs, which is also quite essential to this area. It is generally acknowledged that prioritizing the fault-sensitive test cases is a core task in software testing, because not any arbitrary test case can detect failures, and hence executing test cases with higher fault-sensitivity as early as possible helps save the test cost. In fact, this saving can be even more important in MT for DL software. It has been demonstrated that MT can help to avoid the high cost of oracle preparation for individual test cases in traditional DL testing [39], and hence makes the test executions with abundant MPs become possible [4, 5]. As a result, due to the general high execution cost of MPs in many DL application domains, prioritizing fault-sensitivity MPs should be seriously considered, without which the revealing of some detected violations **can be greatly delayed**. And there may even exist a few violation-inducing MPs being left not executed, and thus conceal

certain critical defects. Therefore, we argue that, besides exploring new fields and MRs, we should also take great notice of the efficiency of MT for DL testing. In other words, a prioritization method is necessary to select fault-sensitive MPs and boost the fault detection, i.e., the revealing of detected violations, in MT for DL testing.

Actually, researchers in the field of MT have found that *the MPs whose source and follow-up cases imply high diversity of execution tend to have a high probability to reveal violations in general* [7, 8]. Thus, it should be quite helpful for DL testing to prioritize the fault-sensitive MPs based on their diversity of execution, so as to obtain better violation revealing efficiency. For the traditional software, the diversity can be measured with some metrics on the white-box information, e.g., the difference on branch coverage in executing the source and follow-up cases [3]. But these metrics are not suitable for DNNs. And though there are some neuron coverage criteria proposed for DNNs [23, 25], they are mainly designated to measure the adequacy of test suits, thereby may not well adapt to model the execution diversity of MPs. As a result, an effective diversity metric is lacked and desired to select fault-sensitive MPs in DL testing.

To tackle the above problems and boost the revealing of detected violations in DL testing, in this paper, we propose a new type of diversity metric on DNN executions and build a method to prioritize the MPs based on this metric. Specifically, our diversity metric models the execution of DNNs based on the internal status (i.e., neuron outputs) of DNNs. Given an MP, our metric regards the output distribution discrepancy of some neurons for the source and follow-up cases in this MP as its diversity. More importantly, our prioritization method breaks the DNN to test into two parts, where the first part includes the neuron layers from the first layer to the *break point*[1]; while the latter part includes the neuron layers from the break point to the last layer. When performing testing, it firstly executes all MPs on the first part of the tested DNN. Next, it prioritizes the MPs based on their execution diversity calculated on the first part of the tested DNN. Finally, it follows the priority to execute the MPs on the latter part of the tested DNN and inspect their output relation against MRs. As a result, the MPs with higher diversity will be granted higher priority to be executed and inspected, with the hope of revealing violations and exposing defects more efficiently.

Such an expectation has been verified by our empirical study. We evaluate the proposed diversity metric and prioritization method by applying them to test the DNNs for the representative image classification task. The result shows that the proposed method and metric can effectively select and prioritize the fault-sensitive MPs to boost the revealing of detected violations on three typical image classification DNN models. They significantly outperform the setup of no prioritization and prioritization with the neuron coverage-based diversity metrics. Besides, we also study the configuration sensitivity of our method and provide advice on the choice about the location of break point, types of extracted neuron outputs, and volume of adopted information. Finally, we found that our method and metric can effectively prioritize the MPs of different MRs. Based on the above observations, we further discuss the potential of our diversity metric to benefit the selection and design of effective MRs. The replication package and more specific results can be found in our online artifact [1].

In summary, this work makes the following contributions:

- We propose a diversity-guided method to select and prioritize the fault-sensitive MPs of DL software to boost the revealing of detected violations in DL testing. To the best of our knowledge, this is the first attempt on prioritizing MPs to improve the efficiency of MT for DL testing.
- We propose a new diversity metric for the MPs of DNNs based on the distribution discrepancy of neuron outputs. We implement this metric based on four widely-used distribution discrepancy measurements and discuss their performance.
- We perform an extensive experiment to evaluate the proposed diversity metric and prioritization method. The result shows that using our method and diversity metric can effectively and stably boost the revealing of detected violations in DL testing. We also found that our metric outperforms the baseline metrics based on neuron coverage with large margins and may help to select and design effective MRs.

The rest of the paper is structured as follows. Section 2 illustrates the background about the execution of DNNs and the use of MT in DL testing. Section 3 introduces the intuition from MT for traditional software and our preliminary exploration that both inspire our methodology. Then, Section 4 elaborates our prioritization method and diversity metric, as well as the MRs and test objects for evaluating our method. Sections 5 and 6 describe the settings and the results of the experiments to evaluate the proposed method, respectively. Next, Section 7 discusses some potential value of our proposal in MR design and selection and the threats to validity. Section 8 presents some related works. Finally, Section 9 draws a conclusion and gives ideas on our future work.

2 BACKGROUND

2.1 Deep Neural Network and Its Execution

Traditional software is made up of the code statements, which next constitutes coarse-grained units like the methods and classes. When executing the traditional software, each code statement is executed one by one according to the logic designed by programmers. Differently, DL software is powered by the deep neural networks (DNNs). When executing the DL software, the input is first represented in a float form and next transformed to the output according to the aggregated effects of the computation on neurons in its DNN [23].

As shown in Figure 1, the interconnected *neurons* are generally regarded as the constitutive unit for DNNs. Each neuron performs linear, convolutional, recurrent, or the other calculation to extract distinct information from its float input. The raw calculation result can be further processed by the activation function, normalization, and other post-processors to form the final output of this neuron. The neurons are organized into one input *layer*, one output layer, and one or multiple hidden layers. Multiple layers can further constitute a *block* as a coarse-grained functional unit. Several neuron layers are stacked to formulate a path. One neuron layer (e.g., Layer 3) takes the outputs of the neurons in its precedent layer (e.g., Layer 2) as input. In some advanced DNNs, a block can contain multiple parallel paths. For example, ResNet adds a parallel residual shortcut to directly bridge the input and output of its residual blocks [16].

[1] In this paper, we refer to the neuron layer along which the tested DNN is split into two parts as the "break point". The detailed definition is given in Section 4.1.

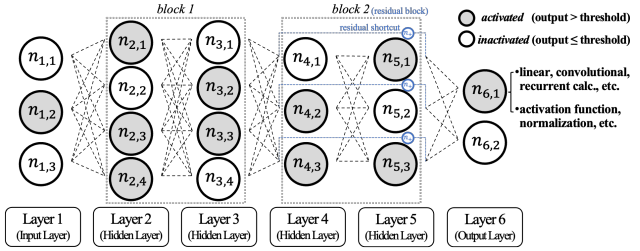


Figure 1: Illustration of A Deep Neural Network

When executing a DNN with a specific test case, the DNN calculates the neuron outputs layer by layer based on the topology of the layers, until the results of the output layer are obtained to formulate a final prediction (e.g., a classification). The neurons in each layer are not connected to each other, thus are usually calculated simultaneously with the parallel calculator like GPUs. In software testing, it is essential to measure the internal status (e.g., statement coverage) of the software under an execution. Several studies on DL testing criteria have found that the neuron outputs can depict the status of neurons (e.g., activated or not) under distinct executions, which help to measure the adequacy of test suites [23, 25].

2.2 Metamorphic Testing for Deep Learning

DL testing is known to face the oracle problem, as it requires very laborious manual annotation to obtain the oracle (i.e., label) for the individual test case of DL software. Metamorphic Testing (MT) is a solution to bypass the oracle problem [6], thus becomes a quite popular technique in DL testing [5]. As introduced in Section 1, given a pile of test cases (also known as the source cases), MT does not inspect the correctness of their individual outputs. Instead, it constructs several Metamorphic test case Pairs (MPs) with some Metamorphic Relations (MRs) and checks if the outputs for the source and follow-up cases in each MP satisfy the specified MR. For example, when we want to identify the incorrectness of each individual output of a handwritten digit image recognition model, the manually annotated labels are required as reference. But using MT, we can define an MR to properly shift the image to generate the follow-up case for each source case to form several MPs. Afterward, it is straightforward to inspect if the outputs of the two cases are equivalent. If this equivalent MR is not satisfied, a violation presents to reveal that at least one output is erroneous. Due to the strengths of MT in enabling the usage of unlabeled data and enriching the DL test executions [5], many studies propose various MRs, notably the equivalent MRs, and have succeeded in revealing much critical misbehavior in various DL software [4, 5, 17, 31, 32, 35–37, 39].

3 PRELIMINARIES

3.1 Intuition from MT for Traditional Software

As introduced in Section 1, in this work, we aim to prioritize the MPs that are more like to trigger violations in DL testing. Though no existing work pays attention to the prioritization of MPs in DL testing, there have been some findings on the correlation between the features of MPs and their defect revealing probability on traditional software. For instance, by investigating the execution of MPs on the shortest/critical path programs, Chen et al. [8] propose the intuition that the MPs whose test cases lead to diverse executions are more likely to trigger violations. They later add that such diversity can cover all aspects of program executions, such as the sequence of statements executed and the values taken by variables [7]. Subsequently, Cao et al. [3] design effective diversity metrics based on the execution profiles to fulfill this intuition.

This intuition could be reasoned by considering the sufficiency and diversity of testing. Specifically, given the test cases t_s and t_f in an MP, consider that t_s and t_f lead to diverse executions. Supposing that t_s touches a faulty execution pattern, e.g., entering a faulty execution path in the tested traditional software, and obtains an incorrect output, then t_f is more likely to bypass similar faulty execution pattern and gets a different (usually a correct) output. As a result, two outputs tend to violate the corresponding MR and the defect is thus revealed. By contrast, if t_s does not touch a faulty execution pattern, t_f then tries to examine the other unchecked execution patterns to explore the potential defects. This also increases the probability of exposing defects with this pair of test cases.

3.2 Diversity Measurement on DNN Executions

We follow the above intuition to prioritize MPs in DL testing with the guidance of execution diversity. We have preliminarily investigated some naive diversity metrics based on the existing neuron coverage criteria about their abilities in distinguishing the neuron activities on different MPs [40]. But since the neuron coverage criteria mainly consider a few fixed neuron "activation states", these metrics are not fine-grained enough to depict the internal execution status of DNNs. Therefore, in this work, we try to model the difference in the neuron outputs in a finer granularity for more precise internal execution status representations. Specifically, **we propose a new type of diversity metric** that measures the distribution discrepancy between the neuron outputs for the two test cases in the specified MP as the diversity of this MP. In this way, the MPs whose two test cases have a larger neuron output distribution discrepancy may inspect more diverse DNN execution patterns, e.g., relying on more dissimilar features, thus are more apt to touch and expose defects.

This proposal is quickly supported by our preliminary exploration on a VGG16 image classification model[2]. Specifically, we design MRs to construct MPs by changing the style of images and expect the predicted object species to be identical. After studying the neuron outputs for some MPs, we found that the MPs that violate MRs do cause a fairly larger difference on the neuron output distribution in general. Here we give an example to illustrate this finding. To simplify the demonstration, we pick two MPs that share the same source case s but have distinct follow-up cases f_1 and f_2. The MP made up of s and f_1 passes the test; while the other MP made up of s and f_2 triggers a violation. We analyze the neuron outputs of the first, middle, and last convolutional layers in the tested VGG16 model. Figure 2 shows the output distribution of ten randomly picked neurons from each of these layers for s, f_1, and f_2. We can see that, on all the three layers, there is a fairly larger discrepancy between the neuron output distribution of s and f_2 than that of s and f_1. This demonstrates that the **violated MP** made up of s and f_2 would **imply a larger discrepancy** in the neuron output distribution. Besides, from Figure 2, we also notice the range of neuron outputs can vary a lot. In this case, the coarse neuron

[2]This preliminary study also uses the setup for the full evaluation of our prioritization method and diversity metric (c.f., Section 5). And we also fully evaluate our method and metric on VGG16 in our evaluation, whose results will be discussed in Section 6.

output representations, e.g., activation states divided by few thresholds, can conceal some discrepancy to identify the violated MPs. This also confirms our need to precisely model neuron outputs.

Overall, these findings show the possibility to use our diversity metric based on the distribution discrepancy of neuron outputs to effectively discriminate and prioritize the fault-sensitive MPs that are apt to trigger. Actually, they also further suggest the outputs of some neurons (e.g., neurons in a layer) could be informative enough to discriminate the fault-sensitive MPs. In the following, we build our prioritization method based on the proposed diversity metric and design the specific efficient implementation of this metric.

4 METHODOLOGY

4.1 Overview of Prioritization Method

Based on the diversity metric proposed in Section 3.2, we build a prioritization method to boost the revealing of detected violations in DL testing. Consider that, after MRs are defined, the major cost of MT usually lays on the execution of MPs. Once the MPs are executed on the *whole* DNN, the test outputs can be easily obtained and checked against MRs. Therefore, **we propose to divide the execution of a DNN into two parts and prioritize the execution of MPs on one part based on their execution on the other part.** Specifically, we pick a neuron layer as the *break point* and break the DNN under test along this break point into two groups of disjoint neuron layers. We use a layer as the division unit considering that the neurons in a layer are usually calculated in a parallel way. Next, we execute *all* MPs on one group of layers and calculate the execution diversity for all MPs. Afterward, we execute the MPs *one by one* on the other group of layers according to the previously obtained execution diversity of each MP. As a result, the MPs with high execution diversity, which are regarded as fault-sensitive, will be completely executed on all the neuron layers at first. Their test outputs will thus be first obtained for inspection against MRs to efficiently reveal defects. In this work, we naturally divide the neuron layers according to their topological order regarding that a neuron layer cannot be executed until its precedent layers are executed.

More specifically, let us denote the DNN to be tested as $\mathcal{N} = \{L_1, L_2, \ldots, L_{|\mathcal{N}|}\}$, where L_i is the i-th neuron layer in $\mathcal{N}$ and $|\mathcal{N}|$ is the total number of the neuron layers in $\mathcal{N}$. And we denote a pile of available MPs for $\mathcal{N}$ as $\mathbb{MP} = \{MP_1, MP_2, \ldots, MP_{|\mathbb{MP}|}\}$, where MP_i is the i-th MP and $|\mathbb{MP}|$ is the total number of MPs. Figure 3 shows the overall process of our method in boosting the testing for $\mathcal{N}$ on $\mathbb{MP}$. Firstly, we pick a neuron layer L_k as the break point and break $\mathcal{N}$ into two parts (step 1). The first part of $\mathcal{N}$ includes the neuron layers from the first layer to the break point (colored in green), denoted as $\mathcal{N}_A = \{L_1, L_2, \ldots, L_k\}$. And the second part of $\mathcal{N}$ includes the neuron layers after the break point to the last layer (colored in blue), denoted as $\mathcal{N}_B = \{L_{k+1}, L_{k+2}, \ldots, L_{|\mathcal{N}|}\}$. Next, we execute all MPs in $\mathbb{MP}$ on $\mathcal{N}_A$ and measure their execution diversity (step 2). After that, we sort all MPs in $\mathbb{MP}$ based on their diversity calculated on $\mathcal{N}_A$ in descending order and obtain a prioritized sequence $\mathbb{MP}' = \left\langle MP'_1, MP'_2, \ldots, MP'_{|\mathbb{MP}|} \right\rangle$, where $MP'_i \in \mathbb{MP}$ is the i-th MP to execute on $\mathcal{N}_B$ (step 3). The MPs that are considered as more likely to trigger violations are thereby prioritized. Finally, we execute the MPs in $\mathbb{MP}'$ one by one on $\mathcal{N}_B$ to obtain the final prediction outputs for their source and follow-up cases and inspect the outputs against MRs to reveal violations (step 4).

4.2 Implementation of Our Diversity Metric

Aiming to boost the revealing of detected violations, a both accurate and cheap implementation of the proposed metric is desired. For a DNN $\mathcal{N}$ that is broken into $\mathcal{N}_A$ and $\mathcal{N}_B$ at L_k, given an MP, we use the distribution discrepancy between the neuron outputs of L_k for its two test cases to represent its execution diversity on $\mathcal{N}$. Specifically, let us denote the given MP as $MP = (t_s, t_f)$, where t_s and t_f are the the source and follow-up cases in MP, respectively. And we denote the normalized neuron outputs of L_k for t_s and t_f as $o(t_s, L_k)$ and $o(t_f, L_k)$, respectively. The diversity of MP is thereby calculated as:

$$\text{Diversity}(MP) = \text{Discrepancy}\left(o(t_s, L_k), o(t_f, L_k)\right)$$

Here we only use the neuron outputs of L_k considering that they are informative for discriminating the execution on $\mathcal{N}$. On one hand, since the outputs for the neurons in a layer are calculated based on the neurons on previous layers, the outputs of the neurons in the break point L_k might have efficiently summarized the execution on $\mathcal{N}_A$. On the other hand, since $\mathcal{N}_B$ takes the output of $\mathcal{N}_A$ as input, the different outputs of L_k are also apt to cause diverse executions on $\mathcal{N}_B$. This choice helps to save the cost on diversity calculation. As a reminder, it is much cheaper than the execution of DNNs.

There are several measurements for the distribution discrepancy. But considering the inexplicability of DNNs, we do not determine the most effective implementation of our diversity metric from theory. Instead, we implement our metric with several representative measurements and compare their performance in our evaluation. Specifically, we consider four common measurements, namely the Kullback-Leibler divergence, Jensen-Shannon divergence, Wasserstein distance, and Hellinger distance. Kullback-Leibler divergence (KL) calculates how one distribution is different from another reference distribution. The diversity metric implemented based on KL measures the difference from the neuron outputs of L_k for follow-up case to that for source case as:

$$\text{Diversity}_{KL}(MP) = \sum_{n_{k,j} \in L_k} o(t_s, n_{k,j}) \ln \frac{o(t_s, n_{k,j})}{o(t_f, n_{k,j})}$$

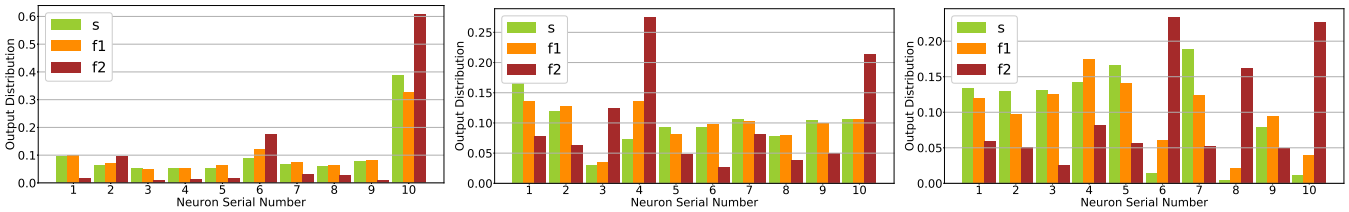
where $o(t_s, n_{k,j})$ and $o(t_f, n_{k,j})$ are the outputs of the j-th neuron $n_{k,j}$ in L_k for t_s and t_f, respectively. Jensen-Shannon divergence (JS) is an improved version of KL to measure the difference of two distributions in a symmetric way. The diversity metric implemented based on JS is calculated as:

$$\text{Diversity}_{JS}(MP) = \frac{1}{2}\left(\text{D}_{KL}(o(t_s, L_k) \| M) + \text{D}_{KL}(o(t_f, L_k) \| M)\right)$$

where $M = \frac{1}{2} \cdot \left(o(t_s, L_k) + o(t_f, L_k)\right)$ and $\text{D}_{KL}(\cdot)$ is the KL distance. Wasserstein distance (WD) calculates the minimum cost to turn one distribution into another distribution. We also implement the proposed diversity metric based on WD, which is calculated as:

$$\text{Diversity}_{WD}(MP) = \inf_{\gamma \in \prod(o(t_s, L_k), o(t_f, L_k))} E_{(x,y)\sim\gamma}\left[\|x - y\|\right]$$

And we also implement the proposed diversity metric based on the Hellinger distance (HD), which measures the difference between



(a) Output Distribution at the First Convolutional Layer (b) Output Distribution at the 7th Convolutional Layer (c) Output Distribution at the Last Convolutional Layer

Figure 2: Output Distribution of 10 Neurons in the First, Middle, and Last Convolutional Layers of VGG16 for Three Test Cases

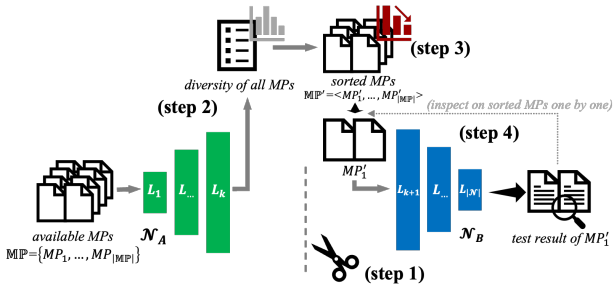


Figure 3: Overview of Prioritization Method

two distributions with Hellinger integral. The HD-based implementation is calculated as:

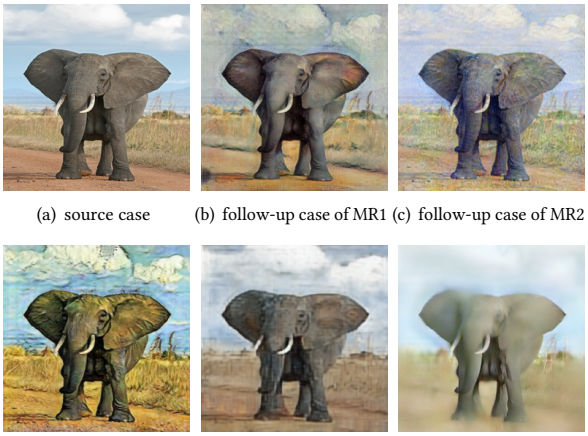
$$\text{Diversity}_{HD}(MP) = \frac{1}{\sqrt{2}}\sqrt{\sum_{n_{k,j}\in L_k}\left(\sqrt{o(t_s, n_{k,j})} - \sqrt{o(t_f, n_{k,j})}\right)^2}$$

4.3 MRs for Image Classification Models

Considering the image classification is an essential DL task and has been used as the representative subject in many studies on DL testing criteria [23], in this work, we also explore the usefulness of our method in boosting the revealing of detected violations on the image classification models. Given an image (e.g., Figure 4(a)), the model predicts the species of the major object in the image (e.g., elephant). To obtain diverse and vivid images, we do not use the MRs of simple affine transformations like shift and shear. Instead, we use five typical MRs to transfer the images into diverse styles following the idea in classical studies [35, 39]. All these MRs require the source and follow-up outputs to be identical because the species of the major object in the image is not affected. The style transfer for the input images is achieved with the popular CycleGAN technique [41] for good transferring quality. The five MRs are as follows[3]:

- **MR1: Cezanne.** This MR generates a follow-up case that describes the content of the source case using the Paul Cezanne painting style. Figure 4(b) is an example of the follow-up case generated by this MR.
- **MR2: Monet.** This MR generates a follow-up case that describes the content of the source case using the Claude Monet painting style. Figure 4(c) is an example of the follow-up case generated by this MR.

[3]These MRs are chosen to generate fairly vivid images as high-quality follow-up cases. We do not mean the usefulness of our method is restricted to these MRs, as well as the image classification task, which are both taken as representative subjects in this work. We will keep further trying to use our proposal with more subjects in our future work.



(a) source case (b) follow-up case of MR1 (c) follow-up case of MR2

(d) follow-up case of MR3 (e) follow-up case of MR4 (f) follow-up case of MR5

Figure 4: Illustration of the Test Cases for Five MRs

- **MR3: VanGogh.** This MR generates a follow-up case that describes the content of the source case using the Vincent van Gogh painting style. Figure 4(d) gives an example of the follow-up case generated by this MR.
- **MR4: Reconstruction.** This MR compresses the source case into a low-dimensional representation and then reconstructs it to obtain a follow-up case with a little difference to the source case. Figure 4(e) is an example of the follow-up case generated by this MR.
- **MR5: DSLR.** This MR generates a follow-up case that simulates the DSLR photo with a shallower depth of field for the content of the source case. Figure 4(f) is an example of the follow-up case generated by this MR.

5 EXPERIMENTAL SETUP

5.1 Research Questions

We evaluate the usefulness of our proposal in boosting the revealing of detected violations against the following research questions:

RQ1: Can our method effectively boost the revealing of detected violations?

In this RQ, we analyze the overall effectiveness of the proposed diversity metric and prioritization method by comparing their performance to the baselines in testing three widely-used image classification models. Due to the space limit, we choose a layer at the middle of each tested DNN as the break point and adopt the neuron

outputs processed by the activation function for measuring diversity by default here[4]. The reason to choose such configuration as the default setup is discussed in RQ2. *In fact, our method and metric can also effectively outperform the baselines under the other setups.* More comparison under the other setups can be found at [1].

RQ2: How does the method configuration affect the boosting effectiveness?

The location of break point, type of neuron outputs, and volume of neuron outputs are three key parameters in diversity measurement. In this RQ, we study the configuration sensitivity of our method against these parameters.

RQ3: Under different MRs, how effective is our proposal and what is the distribution of diversity among MPs?

In RQ1 and RQ2, we merge the MPs of all MRs as a whole. In this RQ, we first investigate if our method can effectively prioritize the MPs of different MRs. Next, we characterize the diversity among the MPs of different MRs; in particular, we study the correlation between the fault detection ability of MRs and the diversity of their MPs. Here we also use the default parameter configuration in RQ1.

5.2 Test Cases and Test Objects

To test the image classification models with MT, we collect source cases from a classical image classification benchmark, the ImageNet ILSVRC 2012 dataset [20]. It includes the images of several common objects in daily life. We randomly pick 1000 images from the test set of ImageNet ILSVRC 2012 as the source cases. For each source case, we use the proposed MRs to construct five corresponding MPs. Finally, we obtain a total of 5000 MPs in random order.

And we introduce three classical image classification models as the test objects, i.e., VGG16 [28], Inception V3 [33], and ResNet50 [16]. VGG16 includes 5 simple blocks, each of which contains several linearly stacked neuron layers with small-size convolutional kernels. Inception V3 includes 11 Inception blocks. An Inception block contains multiple parallel paths, each of which stacks independent convolutional neuron layers. In our evaluation, we place the break point in the longest path of each block. And ResNet50 is made up of 16 residual blocks. A residual block stacks some convolutional neuron layers and simultaneously uses a shortcut path to transfer information from the input to the output of this block. Keras [34] has released its officially implemented and pretrained VGG16, Inception V3, and ResNet50 models for the ImageNet ILSVRC 2012 task. We load these official models as the test objects.

5.3 Baseline Diversity Metrics

For traditional software, the difference between the code coverage of two test cases in an MP is usually taken as its diversity [3]. Zhang and Xie [40] preliminarily studied five types of neuron coverage-based diversity among the MPs for DNNs. We take this idea to use those neuron coverage-based diversity metrics as the baselines[5]. The five baseline metrics are ΔNC, ND, SD, BD, and TD.

[4]For the officially implemented VGG16, Inception V3, and ResNet50 models released by Keras, we adopt the outputs of the neuron layers named *block4_conv1* (activated outputs), *activation_55*, and *activation_25* to calculate the diversity, respectively.

[5]There are other DL testing criteria like DeepGini [13] and Surprise Adequacy [19]. But they aim to select **individual test cases** and usually rely on the **final prediction**. Thus, they cannot be easily adapted to prioritize **MPs** based on the **partial DNN**.

ΔNC measures the difference in the ratios of activated neurons in L_k under two DNN executions. Specifically, a neuron is considered as activated when its output is greater than a threshold θ_{NC}. The neuron coverage (NC) is thereby defined as the ratio of the activated neurons to all neurons in the tested model [25]. And ΔNC refers to the difference of NC under the executions of source and follow-up cases in an MP. **ND** calculates the Jaccard distance between the sets of activated neurons in L_k (also known as the Metamorphic Neuron Slices of L_k [40]) under two DNN executions. The activated neurons in ND also refer to the neurons whose outputs are greater than a threshold θ_{ND}. **SD** and **BD** identify the uncovered neurons based on the k-multisection neuron coverage [23] and measure the ratio of uncovered neurons in L_k under two executions. Specifically, they divide the range of neuron outputs into the major function region and the lower and upper corner-case regions based on the neuron outputs during training phase. The major function region is further divided into k equal subsections. Next, SD regards a neuron as uncovered if its outputs under two executions are within distinct major function subsections. Meanwhile, BD regards a neuron as uncovered if its outputs under two executions are within distinct function regions. And inspired by the top-k neuron coverage in [23], **TD** measures the diversity in the most activated neurons in L_k under two DNN executions. Specifically, it calculates the ratio of the non-overlapped neurons in the k neurons with the greatest outputs under two executions. Following the suggestions in [40], we set θ_{NC}, θ_{ND}, k for SD and BD, and k for TD to be 0.5, 0.75, 2, and 1, respectively, to obtain the optimal discrimination performance.

5.4 Measurement of Prioritizing Performance

The Average Percentage of Faults Detected (APFD) and its variants are the common performance metrics for the test case prioritization methods of traditional software [27]. APFD measures how quickly the specific faults (e.g., faulty statements) can be identified by the prioritized test cases. However, it is hard to identify specific faults in DNNs now. Current DL testing practices mainly aim to reveal the wrong predictions of DNNs [13, 23]. Thus, we elaborate proper metrics to measure how quickly all detected violations can be revealed with the prioritized MPs in $\mathbb{MP}'$. Specifically, we first borrow the idea of APFD to define a new metric called the Average Percentage of Violations Detected (APVD). The only difference between APFD and APVD is that APFD considers the specific faults while APVD cares for the violations that can be detected. APVD is calculated as:

$$\text{APVD}\left(\mathbb{MP}'\right) = 1 - \frac{\sum_{i=1}^{m} TV_i}{n \cdot m} + \frac{1}{2 \cdot n}$$

where n and m are the numbers of all available and fault-sensitive MPs in $\mathbb{MP}'$, respectively, and TV_i is the rank of the i-th fault-sensitive MP in $\mathbb{MP}'$. One higher APVD indicates that the fault-sensitive MPs are better prioritized. And we further scale APVD into the Normalized-APVD (NAPVD) by considering the reachable upper and lower bounds of APVD following Qu et al. [26] and Feng et al. [13]. NAPVD varies within $[0, 1]$ and is calculated as:

$$\text{NAPVD}\left(\mathbb{MP}'\right) = \frac{\text{APVD}\left(\mathbb{MP}'\right) - APVD_{min}}{APVD_{max} - APVD_{min}}$$

where $APVD_{max}$ refers to the APVD when all fault-sensitive MPs are prioritized at first (i.e., being ranked with 1 to m) and $APVD_{min}$ refers to the APVD when all fault-sensitive MPs are placed at last

(i.e., being ranked with $n - m + 1$ to n). The NPAVD of 1 refers to the best prioritization performance. We use NAPVD as the metric of prioritization performance in the evaluation.

6 RESULTS AND ANALYSIS

6.1 RQ1: Overall Effectiveness

6.1.1 Discrimination Ability of Diversity Metrics. In this RQ, we want to first study whether the proposed diversity metric can distinguish the violated MPs from the passed MPs. To this end, we perform two types of non-parametric test [11] to identify the difference between the diversity of the violated and passed MPs. This helps us preliminarily understand the qualitative performance of our diversity metric. Given a specific metric, if there is no significant difference between the diversity of the violated and passed MPs, this metric is considered to be unable to discriminate the fault-sensitive MPs that are apt to trigger violations. Otherwise, we consider this metric potentially helpful in selecting the fault-sensitive MPs.

We first perform the 2-tailed and 1-tailed unpaired Wilcoxon singed-rank test at the σ level of 0.05. The 2nd, 5th, and 8th columns in Table 1 present the test result, where *similar* means that there is no significant difference between the diversity of the violated and passed MPs; while *greater* indicates the violated MPs tend to have higher diversity than the passed MPs w.r.t. the corresponding metric. From the test result, we found that among five baselines (the middle 5 rows), only TD and SD reveal the difference between the violated and passed MPs. ΔNC, ND, and BD cannot reveal such difference on any test object. Meanwhile, the four implementations of our diversity metric (the last 4 rows), i.e., WD, KL, JS, and HD, effectively reveal the difference between the violated and passed MPs on all the three test objects. This result demonstrates that **our metric can discriminate the violated MPs, and the MPs with high diversity w.r.t. our metric tend to trigger violations**.

We also perform the Kolmogorov–Smirnov (K-S) test to quantify the difference between the diversity of violated and passed MPs w.r.t. each diversity metric. The 3rd, 6th, and 9th columns in Table 1 present the test result, where a higher K-S distance implies a larger difference between the violated and passed MPs w.r.t. the corresponding metric. The test result also shows that TD and SD outperform the other baseline metrics. Meanwhile, the four implementations of our diversity metric further outperform TD and SD, where JS and HD show stable supreme discrimination ability over all three objects. This suggests **the potential superiority of our metric, especially being implemented with JS and HD,** in selecting and prioritizing the fault-sensitive MPs.

6.1.2 Performance on Boosting the Revealing of Detected Violations. Next, we study the actual prioritization performance of our method. We use it to prioritize all the 5000 MPs for each test object and record the corresponding NAPVD. We also compare the performance of using different diversity metrics. Given a specific diversity metric, a higher NAPVD indicates that using our method together with this metric can better boost the revealing of detected violations. As a reminder, if multiple MPs have the same diversity w.r.t. a given metric, we sort them based on their original relative order.

The 4th, 7th, and 10th columns in Table 1 present the NAPVD of using different boosting methods and diversity metrics on three test objects. The first line of table body shows the original NAPVD of testing without prioritization (i.e., the MPs are in the initialized random order). Compared to this baseline setup with no prioritization applied, we found that **using our prioritization method can bring a maximum increment of about 0.20 on NAPVD** (values in bold). This confirms the effectiveness of our whole method in boosting the revealing of detected violations in DL testing.

And by further comparing the performance when different metrics are used, we found that our new diversity metric (the last 4 rows) significantly outperform the baseline metrics based on neuron coverage (the middle 5 rows) in general. Specifically, among the five baseline metrics, ΔNC, ND, and BD can hardly boost the revealing of detected violations compared to the setup with no prioritization applied. TD and SD bring some increment of far less than 0.10 on all objects (except SD for ResNet50). The only fairly obvious increment (0.15) is found on SD for ResNet50. Meanwhile, the four implementations of our metric lead to over or nearly 0.10 increment of NAPVD on three objects (except WD for ResNet50). The two implementations based on JS and HD, respectively, bring the most significant increment. Both of them increase the NAPVD on VGG16, Inception V3, and ResNet50 by 0.22, 0.19, and 0.20, respectively.

The above results demonstrate that our diversity metric, notably being implemented with JS and HD, brings much more significant and stable increment on NAPVD than the neuron coverage-based baseline metrics. We consider this is mainly because **our metric models the difference between two executions in a more accurate manner and can well fit different DNNs**. Specifically, the neuron coverage criteria interpret the neuron outputs into some discrete statuses, such as being activated/covered or not. As a result, some potentially helpful fine difference would be ignored if neurons express the same status in multiple executions. Besides, different DNNs may have different output patterns, thus a general threshold may not well recognize the status of neurons across DNNs. Meanwhile, our metric directly measures the distribution discrepancy between the neuron outputs in two executions; thus tends to accurately measure the diversity between executions with sufficient information. And it does not include any sensitive threshold thus can easily generalize to different DNNs. Both strengths may benefit the accurate selection and prioritization of the fault-sensitive MPs.

6.2 RQ2: Configuration Sensitivity

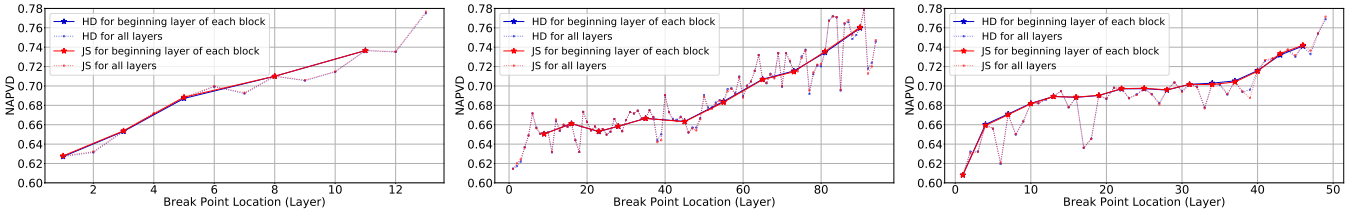
6.2.1 Impact on the Location of Break Point. Given a DNN $\mathcal{N}$, any neuron layer can be used as the break point L_k to divide $\mathcal{N}$ into $\mathcal{N}_A$ and $\mathcal{N}_B$. And the outputs of L_k are used to measure the diversity of MPs on $\mathcal{N}_A$ and prioritize the execution of MPs on $\mathcal{N}_B$. As a result, if k is small, then $\mathcal{N}_B$ is large to save much time of executing the non-fault-sensitive MPs; however, $\mathcal{N}_A$ may offer limited information to discriminate the fault-sensitive MPs. By contrast, if k is big, then $\mathcal{N}_A$ may offer abundant information; while $\mathcal{N}_B$ is small thus the time that can be saved would be reduced. Both situations harm the actual benefit of our method. Thus, we study the performance of placing the break point at different locations to give some advice on the location choice of the break point. Here we also use the outputs processed by the activation function. And we only consider the two most effective implementations, JS and HD, for a clear illustration.

Figure 5 shows the performance of using neuron layers at different locations as the break point on three test objects. Since all

Table 1: Comparison on Overall Effectiveness

Diversity Metric	VGG16 Wilcoxon	VGG16 K-S	VGG16 NAPVD	Inception V3 Wilcoxon	Inception V3 K-S	Inception V3 NAPVD	ResNet50 Wilcoxon	ResNet50 K-S	ResNet50 NAPVD
(None)	-	-	0.4906	-	-	0.4978	-	-	0.4941
ΔNC	*similar*	0.0000	0.4906	*similar*	0.0000	0.4978	*similar*	0.0042	0.4960
ND	*similar*	0.0000	0.4906	*similar*	0.0000	0.4978	*similar*	0.0000	0.4941
BD	*similar*	0.0009	0.4901	*similar*	0.0067	0.5025	*similar*	0.0235	0.5057
TD	*greater*	0.1123	0.5479	*greater*	0.0849	0.5424	*greater*	0.0805	0.5321
SD	*similar*	0.0186	0.4982	*greater*	0.1069	0.5631	*greater*	0.2246	0.6481
WD	*greater*	0.1213	0.5856	*greater*	0.1584	0.6030	*greater*	0.0834	0.5562
KL	*greater*	0.2531	0.6534	*greater*	0.2385	0.6335	*greater*	0.2062	0.5986
JS	*greater*	0.3074	0.7100	*greater*	0.2805	0.6831	*greater*	0.2762	**0.6976**
HD	*greater*	0.3033	**0.7100**	*greater*	0.2758	**0.6841**	*greater*	0.2825	0.6972

A higher K-S distance implies a better ability to distinguish the violated and non-violated MPs. A higher NAPVD implies a better ability to prioritize the violated MPs.



(a) Result on VGG16 (b) Result on Inception V3 (c) Result on ResNet50

Figure 5: Comparison among the Prioritization Performance of Using Neuron Layers at Different Locations as Break Point

three tested DNNs can be firstly divided into multiple blocks, we first compare the performance when using the beginning layer of each block as the break point. The result is shown with the solid lines where each star marks a data point. We found that, on any test object, the performance of both HD and JS generally increases with the increment of the break point location k. This indicates that placing the break point in the latter blocks tends to better prioritize the fault-sensitive MPs than in the previous blocks. And we also present the performance of using any layer as the break point with the dotted lines. We found that, within each block (the dots between two stars), the performance generally increases along with k on VGG16 but fluctuates on Inception V3 and ResNet50. We conjecture this is due to the parallel paths in each block of Inception V3 and ResNet50. As a result, some information may be lost in the outputs of the middle layer in a path of a block; while the beginning layer of each block tends to include more complete information.

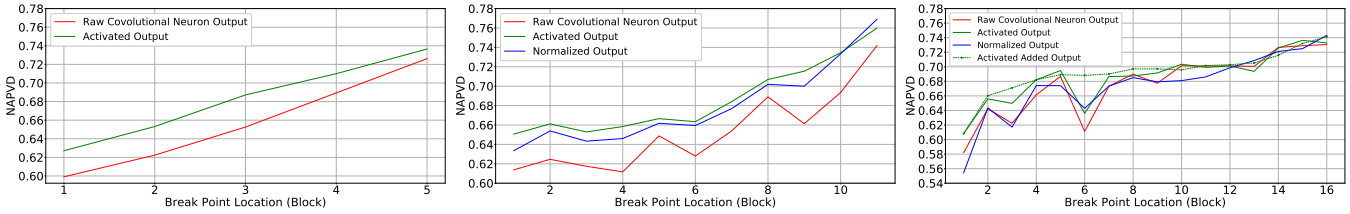
Thus, to balance the accuracy of prioritization and the execution time that can be saved, we consider the neuron layer **at the middle of the tested DNN** as a general choice for the break point. And for the DNNs whose blocks include parallel paths, **the beginning layer of the block at the middle** of DNN is preferred.

6.2.2 Impact on the Adopted Output Type. As introduced in Section 2.1, the neurons in current DNNs usually perform some further processing after perception, such as activation and batch normalization, to better mine the information. Thus, multiple types of neuron outputs can be used to measure the diversity. In this section, we investigate the performance when using different types of outputs. Here we report the results on the beginning layer of all blocks. But we only show the results of HD due to the space limit. The results of JS give a similar conclusion and can be found at [1].

Figure 6 presents the performance of measuring diversity based on different types of outputs on three test objects. For VGG16, there are mainly two types of outputs, namely the raw outputs of convolutional neurons (i.e., the raw outputs of convolutional kernels) and the activated outputs (i.e., the final neuron outputs processed by the activation function). It is clear that using the activated outputs (green line) leads to higher NAPVD than using the raw outputs of convolutional neurons (red line). For Inception V3, there is a new type of outputs, namely the normalized outputs (i.e., the batch-normalized version of the raw outputs). The activated outputs (green line) lead to the best performance in general as well. And ResNet50 contains another new type of activated outputs, the added activated outputs (i.e., the activated outputs of a shortcut addition operation), which merge the information in the ordinary activated outputs with the shortcut paths. Results show that the ordinary activated outputs (green solid line) outperform or are slightly beaten by the raw outputs (red line) and the normalized outputs (blue line). Meanwhile, using the activated added outputs (green dotted line) obtains the best performance in general.

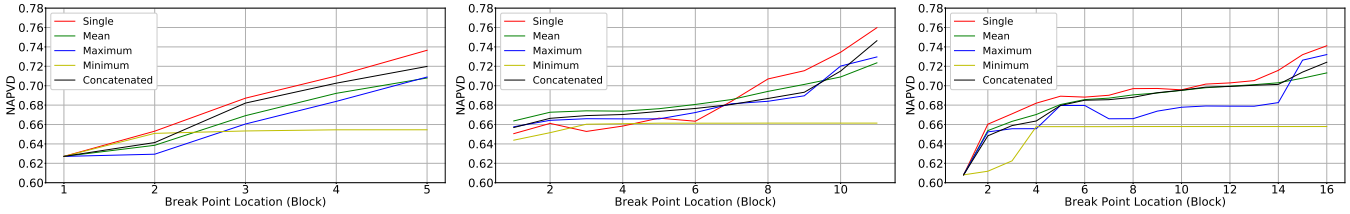
Above results demonstrate that **the activated outputs** lead to the best prioritization performance in general. We consider this is mainly because the activation function shields the useless information about the execution in the raw and normalized outputs. These results suggest us to adopt the neuron outputs being processed by the activation function to accurately measure the diversity between the executions on DNNs. And for the DNNs with some special information branches, such as the shortcuts in ResNet50, **the activated outputs that contain more information** are further preferred.

6.2.3 Impact on the Volume of Information. As introduced in Section 4.2, to save the extra cost on diversity measurement, we only



(a) Result on VGG16 (b) Result on Inception V3 (c) Result on ResNet50

Figure 6: Comparison among the Prioritization Performance of Measuring Diversity based on Different Types of Outputs



(a) Result on VGG16 (b) Result on Inception V3 (c) Result on ResNet50

Figure 7: Comparison among the Prioritization Performance of Measuring Diversity based on Different Volume of Information

measure the distribution discrepancy between the outputs of the *break point layer* for two test cases. In this section, we further study the prioritization performance when using the outputs of *all neuron layers in* $\mathcal{N}_A$ to measure diversity, which tends to provide more information but leads to more extra calculation as well. This helps us understand the necessity to use the information of all neuron layers in $\mathcal{N}_A$ when using our method. And we also use the neuron outputs processed by the activation function and report the results of HD on the layers at the beginning of each block as representative.

We compare the performance of solely using the outputs of break point (denoted as *Single*) with four typical strategies that consider the outputs of all neuron layers in $\mathcal{N}_A$ as follows:

- *Mean:* We calculate the distribution discrepancy on all neuron layers in $\mathcal{N}_A$ and use the average result as the diversity.
- *Maximum:* We calculate the distribution discrepancy on all neuron layers in $\mathcal{N}_A$ and use the maximum result as the diversity.
- *Minimum:* We calculate the distribution discrepancy on all neuron layers in $\mathcal{N}_A$ and use the minimum result as the diversity.
- *Concatenated:* We concatenate the outputs of all neuron layers in $\mathcal{N}_A$ and use the distribution discrepancy for the concatenated result as the diversity.

Figure 7 shows the results. On VGG16 and ResNet50, we surprisingly found that *Single* (red line), which uses less information, outperforms the other four strategies that use more information at any block. And *Single* is also the most effective choice on the 8th to 11th blocks and is only slightly weaker than some other strategies on the first 7 blocks on Inception V3. Such findings suggest that **the neuron output distribution discrepancy of the break point have efficiently represented the execution on $\mathcal{N}$ in general**. At the meanwhile, introducing more information with simple strategies cannot significantly improve the performance. By contrast, it increases the calculation cost of the method and may accumulate noise to harm the prioritization performance.

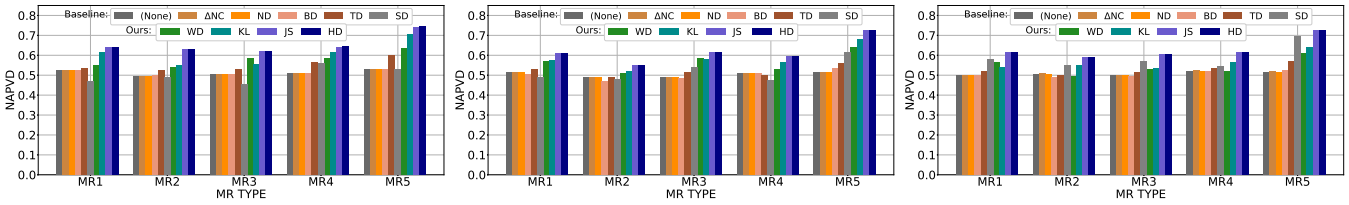
6.3 RQ3: Investigation under Different MRs

6.3.1 Effectiveness Comparison under Each Individual MR. We first compare the NAPVD of our method and metric to the baselines under each MR. This helps us study the effectiveness of our proposal in a finer granularity. Figure 8 shows the comparison result.

We found that using our prioritization method along with any implementation of our diversity metric (the last four bars in each cluster) can increase the NAPVD of no prioritization (the 1st bar in each cluster) under each MR on all test objects. And using the JS-based and HD-based implementations of our metric (the last two bars in each cluster) further outperforms the setup of prioritization with any baseline metric (the 2nd to 6th bars in each cluster) under each MR on all test objects. This means that **the effectiveness of our method and metric under each individual MR is consistent to the overall results reported in RQ1.** It demonstrates that using our prioritization method and diversity metric, notably the JS-based and HD-based implementations, can effectively prioritize the fault-sensitive MPs of different MRs.

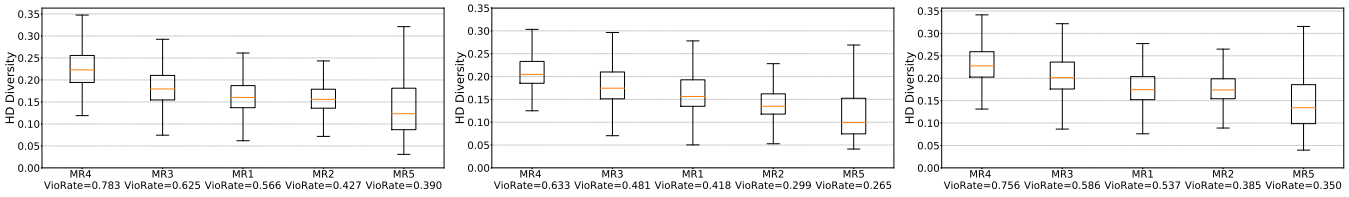
We also study the the performance of using the baseline metrics on each MR. Similar to the overall results in RQ1, ΔNC, ND, and BD can hardly increase the NAPVD of no prioritization on any MR and test object. Meanwhile, in RQ1, TD and SD are found to have certain ability to increase the NAPVD in general. But we found that they cannot work stably on all MRs and test objects. For example, SD is less effective under MR4 than under the other MRs on ResNet50. And both TD and SD even cause negative effect under MR2 and MR4 on Inception V3. As we discuss in RQ1, we consider this is mainly because the mandatory thresholds of these baseline metrics limit their generalizability. By contrast, our diversity metric does not include specific thresholds that may vary across different objects and MRs, thus it has a fairly stable performance.

6.3.2 Distribution of the Diversity Among the MPs of Different MRs. In the above section, we compare the performance of different metrics under individual MR. In this section, we turn to compare the



(a) Result on VGG16 (b) Result on Inception V3 (c) Result on ResNet50

Figure 8: Comparison among the Prioritization Performance of Different Diversity Metrics under Each Metamorphic Relation



(a) Result on VGG16 (b) Result on Inception V3 (c) Result on ResNet50

Figure 9: Box Plots of the HD-Based Diversity among the Metamorphic Test Case Pairs of Different Metamorphic Relations

diversity among the MPs of different MRs to study if our diversity can depict some features of MRs and benefit MT for DL testing.

Previous works suggest that the diversity metric that expresses a correlation to the fault detection ability of MRs may **facilitate the selection of effective MRs** [3, 7, 8]. Inspired by this, we also study the correlation between the fault detection ability of MRs, i.e., the ratio of their MPs triggering violations, and the diversity of their MPs w.r.t. our metric. Consider that we only need to execute MPs on partial tested DNN (i.e., $\mathcal{N}_A$) to obtain our diversity. If such correlation is roughly found, practitioners can efficiently estimate the fault detection ability of the specified MRs with the available source test cases according to the diversity among the MPs of these MRs w.r.t. our metric. They are no longer needed to execute MPs on the whole DNN to analyze the violation ratio. Here we also adopt the HD-based metric as a representative due to the space limit. The box plots for the other implementations can be found at [1].

Figure 9 shows the box plots of the HD-based diversity for the MPs of each MR. We found that the more effective MRs with higher violation detection rates (*abbr.* VioRate) generally have bigger upper quartile, median, and lower quartile on the diversity of their MPs. This demonstrates that the more effective MRs do tend to generate MPs with a higher diversity w.r.t. our metric in general. As a result, practitioners can use some statistical indicators, such as the median, over our HD-based diversity for the MPs, to efficiently estimate the fault detection ability of MRs and **select the effective MRs**.

7 DISCUSSION

7.1 Implication for Designing New MRs

Besides helping to select the effective MRs, a diversity metric that correlates with the fault detection ability of MRs is also potential to **guide the design of MRs** [7, 8]. Specifically, we can summarize the common characteristics among certain MPs with the highest diversity (the MPs of different MRs are not separated) and design new MRs based on the findings. As a result, the new MRs may generate more MPs with high diversity to trigger violations and expose defects. Certainly, practitioners can also study the common features

Figure 10: Example of Designing New MR Based on Diversity

of the actually violated MPs based on the test results. But there are two strengths of using diversity. On one hand, as mentioned in Section 6.3.2, **it is more efficient** to use our diversity metric since it does not require to execute the MPs on the whole DNN. On the other hand, the diversity metric **offers a continuous indicator**, so that we can contrapuntally pick top-k MPs with higher diversity for investigation. In this section, we study the top 3% (150/5000) MPs on VGG16 w.r.t. HD as an exploration. As a reminder, the number of the actually violated MPs (2791/5000) is nearly 20× bigger.

After reviewing the top 3% MPs, we notice that many MPs have two test cases with background of dissimilar hue. The first three columns in Figure 10 list three examples. The 1st and 2nd rows show the source and follow-up cases, respectively. Inspired by this, we design MR_{BG} to mask the background of the source case using a color unlike its background hue from its foreground to obtain the follow-up case. MR_{BG} also expects an identical follow-up output. As an exploration, we manually simulate MR_{BG}. We successfully construct several MPs that have high diversity and trigger violations on VGG16. The last two columns in Figure 10 list two examples. In the first case, VGG16 gives a wrong source output "swab" and a correct follow-up output "tripod". In the other case, VGG16 gives a correct source output "coucal" and a wrong follow-up output "monitor". This trial shows the possibility to efficiently **design new effective MRs** with the help of our diversity metric.

7.2 Threats to Validity

The first threat to validity is about the representativeness of the subjects (i.e., DL task, datasets, models, and MRs) in our evaluation. To counter this threat, we adopt the image classification task, ImageNet dataset, and three classical DNNs, which are all widely used as representative subjects in existing studies on the DL testing criteria [23, 25]. And our MRs are based on the style transfer of images, which is a common idea of MRs for image-related DL software and has been used in many classical studies [35, 39]. Thus, our evaluation is fairly representative. In fact, the rationale of our prioritization method and diversity metric is independent with the DL tasks, models, and MRs. In our future work, we are quite willing to investigate if they will benefit more subjects.

Another threat to validity is on the configuration sensitivity of our method and metric. To counter this threat, we have investigated their performance under different configurations and summarized the general setup. The general setup is found to work effectively and stably on various DNNs and MRs.

The last threat to validity is on the performance metric in evaluation. There is no dedicated performance metric for the prioritization of MPs. We properly adapt the widely-used metrics for the test case prioritization tasks (i.e., APFD and its variants) to formulate a suitable and reasonable metric (i.e., NAPVD).

8 RELATED WORK

Prioritization in MT for Traditional Software. For the testing of traditional software, Chen et al. [8] first reveal the superiority of the MRs that can construct MPs with high execution diversity. They later give some perspectives to measure the execution diversity on traditional software in [7]. Inspired by these, researchers select or optimize MRs based on the accumulative code coverage [2], the coverage distance between test cases [3], and the multiple-level execution diversity [29]. Some studies also adopt this idea to optimize the MP set [12] and select the effective source cases [30]. These studies are mainly based on the coverage information and for the selection and prioritization of MRs or source cases.

In this work, we focus on MT for DL software. We propose a new diversity metric suitable for the MPs of DNNs based on the neuron outputs. And we fulfill the prioritization of fault-sensitive MPs.

Test Case Selection in DL Testing. Many studies propose various criteria to gauge the adequacy of test suits or select fault-sensitive test cases. Pei et al. [25] propose the first testing criterion, neuron coverage (NC), to guide the testing of DNNs. Ma et al. [23] next propose several fine-grained NC criteria to accurately gauge the test cases. By considering the limitation of the coverage-based criteria, later studies design new criteria, such as the Surprise Adequacy [19], DeepGini [13], and so on [24]. Initially, these criteria are mainly used to prioritize the more effective *individual test cases* thus relieve the workload of manually preparing the individual test oracles [24]. Later, a few works take some criteria, such as NC and DeepGini, as objectives to perform the DNN fuzzing [18, 22, 35]. Their aim is to iteratively generate new test cases and the criteria are used to identify the potentially less effective test cases and stop the further mutation on such cases to reduce the *less effective generation.*

Our work differs from them in following aspects. First, we prioritize the fault-sensitive *test case pairs (MPs)* for MT-based methods. As mentioned in Section 1, though MT bypasses the oracle problem, it still faces the selection of fault-sensitive MPs to efficiently reveal defects. This is overlooked by major existing works. Only Zhang and Xie [40] apply several simple neuron coverage-based diversity metrics to preliminarily study the neuron activity of distinct MPs. In this work, we propose a new diversity metric of higher accuracy to identify the fault-sensitive MPs of DNNs and the first prioritization method for such MPs in DL testing. Enhanced with them, MT-based DL testing will not only bypass the costly individual oracle preparation but also reveal the detected defects as early as possible. Besides, our aim is to figure out the execution priority of all the *existing MPs* for *one time* to save the time of executing the non-violated MPs. And to better meet this goal, we design new metric and method rather than use the existing criteria since they usually require the information available after almost the full execution of test cases such as the predicted probability. Moreover, we explore the value of the proposal in helping to select and design effective MRs. All of these further facilitate the efficient DL testing.

9 CONCLUSION AND FUTURE WORK

MT has been widely-used in DL testing to expose defects for various DL software by revealing violations. To boost the revealing of detected violations and expose the defects as early as possible in DL testing, in this paper, we propose the first method to prioritize the fault-sensitive MPs based on the execution diversity of their test cases. To meet this goal, we also devise a new diversity metric for MPs of DNNs based on the distribution discrepancy of neuron outputs. Comprehensive evaluation results on the essential and representative image classification DL task demonstrate our method and metric can effectively and stably boost the revealing of detected violations with significant margins. Our diversity metric is also found to be helpful in the selection and design of effective MRs. In future, we will evaluate our method and metric on more DL tasks and MRs. We will also strengthen our method and metric by considering the features of the other effective DL testing criteria.

ACKNOWLEDGMENTS

This work was partially supported by the National Natural Science Foundation of China under the grant numbers 61972289 and 61832009. And the numerical calculations in this work have been partially done on the supercomputing system in the Supercomputing Center of Wuhan University.

REFERENCES

[1] [n. d.]. The online artifact (including the replication package and more detailed evaluation results) for this paper. https://github.com/imcsq/ASE22-MPPrioritize.

[2] Mahmuda Asrafi, Huai Liu, and Fei-Ching Kuo. 2011. On Testing Effectiveness of Metamorphic Relations: A Case Study. In *Proceedings of the 2011 International Conference on Secure Software Integration and Reliability Improvement, SSIRI 2011, 27-29 June, 2011, Jeju Island, Korea*. IEEE Computer Society, 147–156. https://doi.org/10.1109/SSIRI.2011.21

[3] Yuxiang Cao, Zhiquan Zhou, and Tsong Yueh Chen. 2013. On the Correlation between the Effectiveness of Metamorphic Relations and Dissimilarities of Test Case Executions. In *Proceedings of the 2013 International Conference on Quality Software, Najing, China, July 29-30, 2013*. IEEE, 153–162. https://doi.org/10.1109/QSIC.2013.43

[4] Songqiang Chen, Shuo Jin, and Xiaoyuan Xie. 2021. Testing Your Question Answering Software via Asking Recursively. In *Proceedings of the 2021 IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021*. IEEE, 104–116. https://doi.org/10.1109/ASE51524.2021.9678670

[5] Songqiang Chen, Shuo Jin, and Xiaoyuan Xie. 2021. Validation on Machine Reading Comprehension Software without Annotated Labels: A Property-Based Method. In *Proceedings of the 2021 ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2021, Athens, Greece, August 23-28, 2021*, Diomidis Spinellis, Georgios Gousios, Marsha Chechik, and Massimiliano Di Penta (Eds.). ACM, 590–602. https://doi.org/10.1145/3468264.3468569

[6] T. Y. Chen, S. C. Cheung, and S. M Yiu. 1998. *Metamorphic testing: a new approach for generating next test cases*. Technical Report HKUST-CS98-01. Department of Computer Science, Hong Kong University.

[7] Tsong Yueh Chen, D. H. Huang, T. H. Tse, and Zhi Quan Zhou. 2004. Case studies on the selection of useful relations in metamorphic testing. In *Proceedings of the 2004 Ibero-American Symposium on Software Engineering and Knowledge Engineering*. Citeseer, 569–583.

[8] Tsong Yueh Chen, Fei-Ching Kuo, T. H. Tse, and Zhiquan Zhou. 2003. Metamorphic Testing and Beyond. In *Proceedings of the 2003 International Workshop on Software Technology and Engineering Practice, 19-21 September 2003, Amsterdam, The Netherlands*. IEEE Computer Society, 94–100. https://doi.org/10.1109/STEP.2003.18

[9] Dan C. Ciresan, Ueli Meier, and Jürgen Schmidhuber. 2012. Multi-column deep neural networks for image classification. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*. IEEE Computer Society, 3642–3649. https://doi.org/10.1109/CVPR.2012.6248110

[10] Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: deep neural networks with multitask learning. In *Proceedings of the 2008 International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008 (ACM International Conference Proceeding Series, Vol. 307)*, William W. Cohen, Andrew McCallum, and Sam T. Roweis (Eds.). ACM, 160–167. https://doi.org/10.1145/1390156.1390177

[11] Gregory Corder and Dale Foreman. 2009. Nonparametric Statistics for Non-Statisticians: A Step-By-Step Approach. (01 2009). https://doi.org/10.1002/9781118165881

[12] Guowei Dong, Changhai Nie, Baowen Xu, and Lulu Wang. 2007. An Effective Iterative Metamorphic Testing Algorithm Based on Program Path Analysis. In *Proceedings of the 2007 International Conference on Quality Software (QSIC 2007), 11-12 October 2007, Portland, Oregon, USA*. IEEE Computer Society, 292–297. https://doi.org/10.1109/QSIC.2007.15

[13] Yang Feng, Qingkai Shi, Xinyu Gao, Jun Wan, Chunrong Fang, and Zhenyu Chen. 2020. DeepGini: prioritizing massive tests to enhance the robustness of deep neural networks. In *Proceedings of the 2020 ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, USA, July 18-22, 2020*, Sarfraz Khurshid and Corina S. Pasareanu (Eds.). ACM, 177–188. https://doi.org/10.1145/3395363.3397357

[14] Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. 2013. Speech recognition with deep recurrent neural networks. In *Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*. IEEE, 6645–6649. https://doi.org/10.1109/ICASSP.2013.6638947

[15] Awni Y. Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng. 2014. Deep Speech: Scaling up end-to-end speech recognition. *CoRR* abs/1412.5567 (2014). arXiv:1412.5567

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 770–778. https://doi.org/10.1109/CVPR.2016.90

[17] Pinjia He, Clara Meister, and Zhendong Su. 2020. Structure-invariant testing for machine translation. In *Proceedings of the 2020 International Conference on Software Engineering, Seoul, ICSE 2020, South Korea, 27 June - 19 July, 2020*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 961–973. https://doi.org/10.1145/3377811.3380339

[18] Pin Ji, Yang Feng, Jia Liu, Zhihong Zhao, and Zhenyu Chen. 2022. ASRTest: automated testing for deep-neural-network-driven speech recognition systems. In *Proceedings of the 2022 ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, South Korea, July 18 - 22, 2022*, Sukyoung Ryu and Yannis Smaragdakis (Eds.). ACM, 189–201. https://doi.org/10.1145/3533767.3534391

[19] Jinhan Kim, Robert Feldt, and Shin Yoo. 2019. Guiding deep learning system testing using surprise adequacy. In *Proceedings of the 2019 International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, Joanne M. Atlee, Tevfik Bultan, and Jon Whittle (Eds.). IEEE / ACM, 1039–1049. https://doi.org/10.1109/ICSE.2019.00108

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger (Eds.). 1106–1114.

[21] Minghui Liao, Baoguang Shi, Xiang Bai, Xinggang Wang, and Wenyu Liu. 2017. TextBoxes: A Fast Text Detector with a Single Deep Neural Network. In *Proceedings of the 2017 AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, Satinder P. Singh and Shaul Markovitch (Eds.). AAAI Press, 4161–4167.

[22] Zixi Liu, Yang Feng, and Zhenyu Chen. 2021. DialTest: automated testing for recurrent-neural-network-driven dialogue systems. In *Proceedings of the 2021 ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, Denmark, July 11-17, 2021*, Cristian Cadar and Xiangyu Zhang (Eds.). ACM, 115–126. https://doi.org/10.1145/3460319.3464829

[23] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang. 2018. DeepGauge: multi-granularity testing criteria for deep learning systems. In *Proceedings of the 2018 ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, Marianne Huchard, Christian Kästner, and Gordon Fraser (Eds.). ACM, 120–131. https://doi.org/10.1145/3238147.3238202

[24] Wei Ma, Mike Papadakis, Anestis Tsakmalis, Maxime Cordy, and Yves Le Traon. 2021. Test Selection for Deep Learning Systems. *ACM Trans. Softw. Eng. Methodol.* 30, 2 (2021), 13:1–13:22. https://doi.org/10.1145/3417330

[25] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. In *Proceedings of the 2017 Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*. ACM, 1–18. https://doi.org/10.1145/3132747.3132785

[26] Xiao Qu, Myra B. Cohen, and Katherine M. Woolf. 2007. Combinatorial Interaction Regression Testing: A Study of Test Case Generation and Prioritization. In *Proceedings of the 2007 IEEE International Conference on Software Maintenance (ICSM 2007), October 2-5, 2007, Paris, France*. IEEE Computer Society, 255–264. https://doi.org/10.1109/ICSM.2007.4362638

[27] Gregg Rothermel, Roland H. Untch, Chengyun Chu, and Mary Jean Harrold. 1999. Test Case Prioritization: An Empirical Study. In *Proceedings of the 1999 International Conference on Software Maintenance, ICSM 1999, Oxford, England, UK, August 30 - September 3, 1999*. IEEE Computer Society, 179–188. https://doi.org/10.1109/ICSM.1999.792604

[28] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *Proceedings of the 2015 International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.).

[29] Madhusudan Srinivasan. 2018. Prioritization of Metamorphic Relations Based on Test Case Execution Properties. In *Proceedings of the 2018 IEEE International Symposium on Software Reliability Engineering Workshops, ISSRE Workshops, Memphis, TN, USA, October 15-18, 2018*, Sudipto Ghosh, Roberto Natella, Bojan Cukic, Robin S. Poston, and Nuno Laranjeiro (Eds.). IEEE Computer Society, 162–165. https://doi.org/10.1109/ISSREW.2018.000-5

[30] Chang-Ai Sun, Baoli Liu, An Fu, Yiqiang Liu, and Huai Liu. 2022. Path-directed source test case generation and prioritization in metamorphic testing. *J. Syst. Softw.* 183 (2022), 111091. https://doi.org/10.1016/j.jss.2021.111091

[31] Zeyu Sun, Jie M. Zhang, Mark Harman, Mike Papadakis, and Lu Zhang. 2020. Automatic testing and improvement of machine translation. In *Proceedings of the 2020 International Conference on Software Engineering, ICSE 2020, Seoul, South Korea, 27 June - 19 July, 2020*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 974–985. https://doi.org/10.1145/3377811.3380420

[32] Zeyu Sun, Jie M. Zhang, Yingfei Xiong, Mark Harman, Mike Papadakis, and Lu Zhang. 2022. Improving Machine Translation Systems via Isotopic Replacement. In *Proceedings of the 2022 International Conference on Software Engineering, ICSE 2022, Pittsburgh, USA, 21 May - 29 May, 2022*, Daniela Damian and Andreas Zeller (Eds.). ACM. https://doi.org/10.1145/3510003.3510206

[33] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. 2016. Rethinking the Inception Architecture for Computer Vision.

In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2818–2826. https://doi.org/10.1109/CVPR.2016.308

[34] Keras Development Team. [n. d.]. Keras Deep Learning Framework. https://keras.io/.

[35] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. DeepTest: automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 2018 International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, Michel Chaudron, Ivica Crnkovic, Marsha Chechik, and Mark Harman (Eds.). ACM, 303–314. https://doi.org/10.1145/3180155.3180220

[36] Shuai Wang and Zhendong Su. 2020. Metamorphic Object Insertion for Testing Object Detection Systems. In *Proceedings of the 2020 International Conference on Automated Software Engineering, ASE 2020, Melbourne, Australia, September 21-25, 2020*. IEEE, 1053–1065. https://doi.org/10.1145/3324884.3416584

[37] Yuanyuan Yuan, Shuai Wang, Mingyue Jiang, and Tsong Yueh Chen. 2021. Perception Matters: Detecting Perception Failures of VQA Models Using Metamorphic Testing. In *Proceedings of the 2021 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*. Computer Vision Foundation / IEEE, 16908–16917.

[38] Jie M. Zhang, Mark Harman, Lei Ma, and Yang Liu. 2022. Machine Learning Testing: Survey, Landscapes and Horizons. *IEEE Trans. Software Eng.* 48, 2 (2022), 1–36. https://doi.org/10.1109/TSE.2019.2962027

[39] 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 2018 International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, Marianne Huchard, Christian Kästner, and Gordon Fraser (Eds.). ACM, 132–142. https://doi.org/10.1145/3238147.3238187

[40] Zhiyi Zhang and Xiaoyuan Xie. 2019. On the Investigation of Essential Diversities for Deep Learning Testing Criteria. In *Proceedings of the 2019 IEEE International Conference on Software Quality, Reliability and Security, QRS 2019, Sofia, Bulgaria, July 22-26, 2019*. IEEE, 394–405. https://doi.org/10.1109/QRS.2019.00056

[41] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. 2017. Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks. In *Proceedings of the 2017 IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*. IEEE Computer Society, 2242–2251. https://doi.org/10.1109/ICCV.2017.244