In Defense of Simple Techniques for Neural Network Test Case Selection

Shenglin Bao[*]
School of Computer Science
Fudan University
Shanghai, China
slbao15@fudan.edu.cn

Chaofeng Sha[†]
School of Computer Science
Fudan University
Shanghai, China
cfsha@fudan.edu.cn

Bihuan Chen
School of Computer Science
Fudan University
Shanghai, China
bhchen@fudan.edu.cn

Xin Peng
School of Computer Science
Fudan University
Shanghai, China
pengxin@fudan.edu.cn

Wenyun Zhao
School of Computer Science
Fudan University
Shanghai, China
wyzhao@fudan.edu.cn

ABSTRACT

Although deep learning (DL) software has been pervasive in various applications, the brittleness of deep neural networks (DNN) hinders their deployment in many tasks especially high-stake ones. To mitigate the risk accompanied with DL software fault, a variety of DNN testing techniques have been proposed such as test case selection. Among those test case selection or prioritization methods, the uncertainty-based ones such as DeepGini have demonstrated their effectiveness in finding DNN's faults. Recently, TestRank, a learning based test ranking method has shown their out-performance over simple uncertainty-based test selection methods. However, this is achieved with a more complicated design which needs to train a graph convolutional network and a multi-layer Perceptron. In this paper, we propose a novel and lightweight DNN test selection method to enhance the effectiveness of existing simple ones. Besides the DNN model's uncertainty on test case itself, we take into account model's uncertainty on its neighbors. This could diversify the selected test cases and improve the effectiveness of existing uncertainty-based test selection methods. Extensive experiments on 5 datasets demonstrate the effectiveness of our approach.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**.

KEYWORDS

test case selection, deep learning, k-nearest neighbor

[*]Also with East China Institute of Computing Technology
[†]Corresponding Author

ACM Reference Format:
Shenglin Bao, Chaofeng Sha, Bihuan Chen, Xin Peng, and Wenyun Zhao. 2023. In Defense of Simple Techniques for Neural Network Test Case Selection. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '23), July 17–21, 2023, Seattle, WA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3597926.3598073

1 INTRODUCTION

Deep learning (DL) models or deep neural networks (DNN) are adopted in various application domains. However, they are not easily tested or verified and have recently caused a number of safety-related incidents, such as a serious traffic accident in which Tesla Autopilot mistook a white tractor-trailer for the sky [4]. To address the brittleness of DNNs, testing for DNNs is attracting increasing attention.

As developing a DNN follows a data-centric programming paradigm, the key to DNN testing also lies in the selection of test data. In the DNN software development process data generally needs to be labeled with a large amount of effort. The labeled data can be used to supervise the DNN model training or to test the model before deployment. However labeling the data is a costly task, especially for tasks that require domain-specific knowledge (e.g., medical images, pathology reports, etc.). In most of today's application, collecting a large amount of raw data (e.g., images, text, etc.) for a specific task is a relatively easy and potentially automatic task. These unlabeled data can be used as candidates for DNN model testing, and can be used for improving the model's performance after labeling a subset of instances. To increase the efficiency of DNN testing and detect defects in the model with minimal cost, it is essential to select the test cases with the highest probability of being incorrect and label them accordingly. This ensures that efforts are focused on the most critical aspects of testing, resulting in a higher efficiency ratio.

To improve testing efficiency and reduce labeling cost, a variety of test case selection methods have been proposed for DNN model testing, which include neuron coverage (NC) [30, 31, 39, 44], surprise adequacy (SA) [22, 23, 46], and prediction uncertainty-based [5, 10, 43] methods. Several studies [20, 32, 47] have shown that the test case selection method based on prediction uncertainty achieved the best results and less overhead. Recently, an effective test selection method, TestRank, proposed by Li et al. [29] explores both intrinsic attributes and contextual attributes of test samples. Specifically, TestRank trains a ranking model integrating a graph convolutional network (GCN) and a multi-layer Perceptron (MLP) to estimate the probability of test samples being incorrectly predicted by the target DNN model. As demonstrated in their experiments, TestRank has achieved the best results among the current DNN test case selection methods. However, training a GCN is a complicated option which could be replaced by a simple design.

In this paper, we aim to improve the effectiveness of simple test selection methods based on prediction uncertainty [5, 10, 43]. The prediction uncertainty of DNN models (e.g., DeepGini, Entropy, etc.) is obtained from the probability distribution estimated by the testing DNN model. However, several studies have shown that the prediction confidence of the modern DNN models might be not accurate enough [15, 25]. Un-calibrated DNN models could suffer from over-confidence or under-confidence, i.e., they give high probability to wrong predictions or low probability to correct predictions [13, 37, 40]. On one hand, for a single test case, the uncertainty calculated in terms of the original prediction probability distribution of the DNN model might also deviate significantly from the actual one, thus affecting the effectiveness of the uncertainty-based test case selection methods. In particular, over-confident cases are often easily spared by test case selection methods, thus covering up the errors. On the other hand, the uncertainty of a DNN model on the test case could be induced from its neighbors. Figure 1 shows some over-confidence and under-confidence examples tested by the ResNet-18 DNN model on CIFAR-10 dataset. As shown in the figure, there is a significant disagreement in the probability distribution between the samples that cause over/under-confidence and the neighboring samples with similar features, regardless of whether their prediction results are correct or not.

To exploit the locality of DNN model's prediction, we propose a test case selection method based on k-nearest neighbor (k-NN) prediction smoothing: Nearest Neighbor Smoothing (NNS). Concretely, we adjust the prediction distribution on a test case by interpolating its original distribution and that of its neighbors. NNS is a plug-and-play technique that can be easily integrated into existing uncertainty-based test case selection methods. We applied NNS to four popular uncertainty-based test case selection methods and evaluated them extensively on five datasets. We experimentally compare NNS with its original uncertainty-based counterparts, and the results show a significant improvement over the original methods. Compared to TestRank, NNS is lightweight and achieves comparable or even better performance. Although SA-based methods demonstrate some advantage on adversarial examples, NNS shows a more pronounced advantage on clean data. The efficiency of NNS is more than an order of magnitude higher than TestRank and SA-based methods.

Our main contributions can be summarized as follows:

- We propose a DL test case selection method named NNS based on k-NN prediction smoothing.
- We instantiate NNS with two representation learning methods and integrate it into 4 uncertainty-based test selection methods.
- We empirically evaluate our proposed method NNS against TestRank, SA-based and uncertainty-based test selection methods.

2 BACKGROUND AND RELATED WORK

We introduce the background of DNN, testing techniques for DL system, TestRank, and k-NN for DL.

2.1 Deep Neural Networks

Deep Neural Networks (DNNs) are the foundation of deep learning. A DNN model consists of multiple layers, each of which contains several small computational units called neurons [12]. Each layer in a DNN performs nonlinear computations on the output of the previous layer, which are controlled by learnable parameters of the neurons. The output of each layer in a DNN can be considered as a representation of the input data. The multi-level representation of data can be obtained through learning process of a DNN model. Deeper layers can learn the more abstract features of the input data.

We denote the DNN model as $M(x) = f(\theta, x) : \mathcal{X} \to \mathcal{Y}$, where θ is the model parameter. Assume that the model has L layers, and for the l-th layer, the computation of the layer against the output of the previous layer is denoted as f_l, and the parameters of the layer are denoted as θ_l. The DNN model can be represented as Eq. 1:

$$M(x) = f(\theta, x) = f_{L-1}(\theta_{L-1}, f_{L-2}(\theta_{L-2}, \cdots f_0(\theta_0, x)) \tag{1}$$

Let's take an image classification task as an example, the input of the DNN model is an image $x \in \mathbb{R}^D$. Assuming that there are C classes, the prediction of the DNN model on x is the $y = \arg\max_c p(c|x)$ where $p(c|x)$ denotes the probability predicted by the DNN model M that x belongs to class c.

2.2 Testing for Deep Learning Systems

Testing for deep learning systems has some resemblances with traditional software testing, but at the same time is very different. In the traditional software testing field, **test case selection** and **test case prioritization** techniques have been extensively studied and proven to be very effective testing (especially regression testing) strategies. Test case selection is to select a part of the existing test case set for execution, expecting the selected part of the test cases to detect as many faults as possible. The purpose of test case prioritization is to prioritize the test cases in the test case set in an optimal way, and when the test stops at a certain point, it is expected that the executed test cases will detect as many faults as possible.

In a deep learning classification system, a fault occurs in the DNN software when a DNN model classify a sample differently from its actual class. Unlike traditional software where test cases are written by testers, in deep learning test cases are collected data. In deep learning testing, the test case selection problem can be described as follows: given the target model M for testing, and the budget B, select a subset T_S of test cases from the test suite T, so that as many as faults of M can be detected by T_S, where $|T| \gg B$, $|T_s| = B$.

The current test case selection methods for deep learning are almost all based on test case prioritization, where the test cases in T are ranked by the test case prioritization method, and the first B test cases are selected as T_S. Therefore, later we do not distinguish between test case selection and test case prioritization, we all refer to them as test case selection.

Below we introduce the uncertainty metrics that are used in uncertainty-based test case selection methods, where C is the set

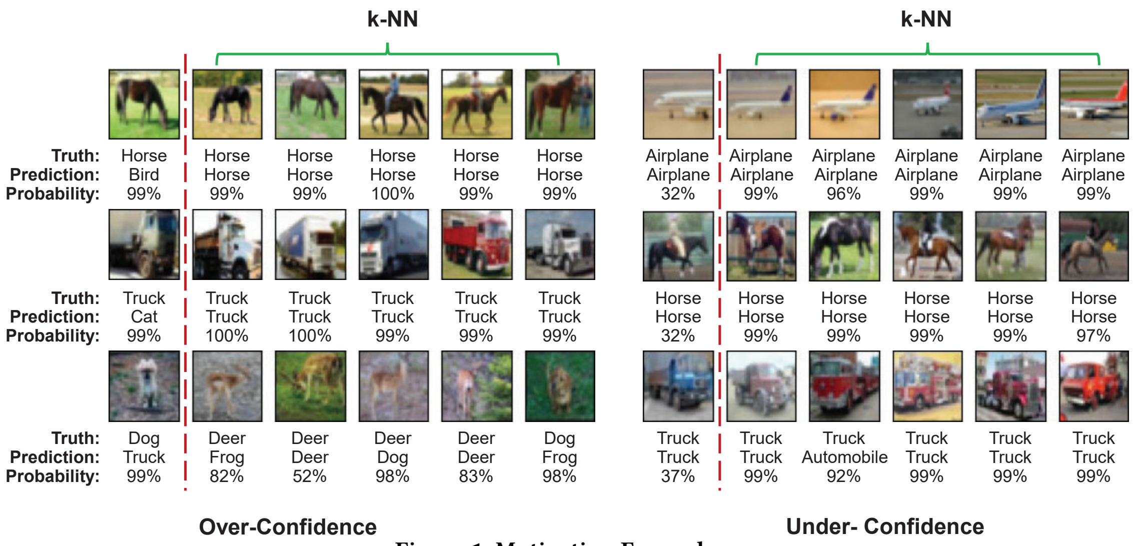

Figure 1: Motivation Examples

of output classes (or labels), and $p_c(x)$ is the prediction probability of x to class c according to M:

- **DeepGini.** The Gini impurity of a test input x with respect to a model M is given by Eq. 2 [10], which measures the likelihood of x being incorrectly predicted. A high Gini impurity represents a high uncertainty.

$$DeepGini(x) = 1 - \sum_{c \in C} p_c^2(x) \tag{2}$$

- **MaxP.** The maximum probability score of a test input x with respect to a model M is given by Eq. 3 [18, 32]. A low maximum probability score represents a high uncertainty.

$$MaxP(x) = \max_{c \in C} p_c(x) \tag{3}$$

- **Margin.** The margin of a test input x with respect to a model M is given by Eq. 4 [42], where $c^* = \arg\max_{c \in C} p_c(x)$. It is a popular and well-regarded data sampling metric in active learning. Intuitively, margin is the difference between the highest and the second highest prediction probabilities. A low margin represents a high uncertainty.

$$Margin(x) = \max_{c \in C} p_c(x) - \max_{c \in C \setminus c^*} p_c(x) \tag{4}$$

- **Entropy.** The entropy of a test input x with respect to a model M is given by Eq. 5 [5], which summarizes the prediction probability distribution. In information theory the entropy is generally represented by H, and it's input is probability distribution. Intuitively, the entropy is lower for a certain prediction where only one $p_c(x)$ is high, and higher for an uncertain prediction where the prediction probability for each class is similar. A high entropy represents a high uncertainty.

$$Entropy(x) = H(p(x)) = - \sum_{c \in C} p_c(x) \log p_c(x). \tag{5}$$

2.3 TestRank

TestRank [29] introduces a ranking model which consists of a Graph Convolutional Network (GCN) and a Multi-layer Perceptron (MLP). In the first stage, TestRank constructs a weighted k-NN graph whose nodes are labeled and unlabeled data instances. Then TestRank leverages a representation learning method (BYOL [14]) to initialize node embedding of the GCN, which is trained in a semi-supervised learning way with the signal whether the tested model correctly predicts the labeled data instance. Finally, the representation learned by GCN concatenated with softmax returned by the trained model is fed into a MLP whose prediction score is employed to rank test case. The benefit of taking test cases' neighbors into account is demonstrated in the experiments of TestRank. However, compared to uncertainty-based test case selection methods, this is achieved by a more complicated design with a GCN and a MLP.

2.4 k-NN for Deep Learning

k-NN [9, 11, 48] is a classical method in machine learning. Deep learning tasks are usually faced with high-dimensional complex data inputs such as images and speech, and retrieving k-NN using raw inputs is no longer effective. Recently, retrieving k-NN via DNN model representations has been shown to be effective in various tasks.

Papernot et al. [38] proposed an indirect estimate of the model prediction confidence by measuring the nonconformity of the predictions of the input data and its nearest neighbors. Jiang et al. [21] proposed a method to calculate the trust score of the DNN model predictions using the topological information of the input data and its nearest neighbors. Bahri et al. proposed to deal with noisy labels using k-NN [2], and to alleviate the predictive churn phenomenon using k-NN [1]. Baldock et al. [3] calculated the prediction depth from the nearest neighbors obtained from the representation of the input data in different layers of the DNN model, and used this to determine the difficulty of the DNN model to learn that data sample.

Our approach also computes k-NN by representation, but we use k-NN for prediction smoothing and apply it to test case selection.

3 METHOD

We first present how to adjust prediction probability distribution with nearest neighbor smoothing, and then describe our selection method.

3.1 Nearest Neighbor Smoothing

We first introduce Nearest Neighbor Smoothing (NNS). The prediction of model M on test x might be inaccurate, because the model is not calibrated or the data is out of distribution. Inspired by the label smoothing technique in machine learning field [45], we smooth the prediction distribution with Eq. 6.

$$p = \alpha p_M + (1-\alpha) p_{kNN}. \tag{6}$$

Here we omit the instance x to avoid clutter. p_M denotes the probability distribution made by the DNN model M. p_{kNN} is the average probability distribution of x's k-nearest neighbors, which will be detailed below. $\alpha \in [0,1]$ is the smoothing weight.

Now we elaborate the designs of p_{kNN}. We retrieve the k-nearest neighbors of instance x, which depends on the representation of instances and the distance measures used. For representation learning, we could use the target DNN model or auxiliary pre-trained models such as SimCLR [7] or BYOL [14]. After learning representation of instances, we use Euclidean or cosine distance to measure the distance between instances. Let $N_k(x)$ denote the retrieved k-nearest neighbors of x, we get the estimated distribution p_{kNN} as follows:

$$p_{kNN}(x) = \frac{1}{k} \sum_{t \in N_k(x)} p_M(t). \tag{7}$$

Here we employ the average of probability distributions of nearest neighbors, while could be weighted according to the distance between neighbors and instance x.

We use the above adjusted prediction distribution to compute the uncertainty metrics such as DeepGini [10] or Entropy [5]. Let us elaborate the benefit of this design. Firstly, we examine the Gini index of an instance x.

$$\begin{aligned}
& \hat{Gini}(x) \\
= \;& 1 - \sum_{c \in C} p_c(x)^2 \\
= \;& 1 - \sum_{c \in C} \left(\alpha p_{M,c}(x) + (1-\alpha) p_{kNN,c}(x)\right)^2 \\
= \;& 1 - \sum_{c \in C} (\alpha^2 p_{M,c}(x)^2 + 2\alpha(1-\alpha) p_{M,c}(x) p_{kNN,c}(x) \\
& + (1-\alpha)^2 p_{kNN,c}(x))^2) \\
= \;& \alpha^2 \left(1 - \sum_{c \in C} p_{M,c}(x)^2\right) + (1-\alpha)^2 \left(1 - \sum_{c \in C} p_{kNN,c}(x)^2\right) \\
& - 2\alpha(1-\alpha) \sum_{c \in C} p_{M,c}(x) p_{kNN,c}(x) + 2\alpha(1-\alpha).
\end{aligned}$$

The first and second terms are Gini index of prediction distributions of x and its k-NNs respectively. Therefore, the overall uncertainty on the test case x takes into account both x and its neighbors. Their contributions are balanced with the weights. Besides, the third term can be seen as the similarity between the test case x with their neighbors in terms of model's prediction distribution (inner product between two distributions). This term could measure the representativeness of x with respect to its neighbors. When we rank test case in terms of this enhance Gini index, those instances whose neighbors have been selected will be penalized as they have been represented. Thus our method could improve the diversity between the selected test cases.

Algorithm 1: Nearest Neighbor Smoothing based Test Case Selection

```
Input: target model $M$, test suite $T$, budget $B$,
       representation extractor $E$, uncertainty measure
       function $measure$, $0 \le \alpha \le 1$, $k > 0$
Output: selected test cases $T^S$
initialize representation list $R = \phi$ ;
initialize prediction list $P = \phi$ ;
initialize distance matrix $DM_{i,j} = 0 (1 \le i, j \le |T|, i \ne j)$ ;
initialize priority queue $Q = \phi$ ;
initialize $T^S = \phi$, $effort = 0$ ;
for $i$ in $|T|$ do
    $R_i = E(T_i)$ ; // representation extraction
    $P_i = M(T_i)$ ; // prediction extraction
end
for $i$ in $|T|$ do
    for $j$ in $|T|$ do
        if $i \ne j$ then
            $DM_{i,j} = CosineDistance(R_i, R_j)$ ; // test
                case distance
        end
    end
end
for $i$ in $|T|$ do
    $p_{kNN} = \phi$ ;
    $s_{kNN} = ArgMin_k(DM_i, k)$ ; // find $k$-NN
    $p_M = P_i$ ;
    for $j$ in $s_{kNN}$ do
        $p_{kNN} = p_{kNN} \cup P_j$ ;
    end
    $p = nns(\alpha, p_M, p_{kNN})$ ; // prediction
        probabilities smoothing
    $uncertainty = measure(p)$ ;
    $insert(Q, < uncertainty, T_i >)$ ;
end
while $effort \le B$ do // test case selection
    $< uncertainty, t >= popMax(Q)$ ;
    $T^S = T^S \cup t$ ;
    $effort = effort + 1$ ;
end
return $T^S$
```

For Entropy we have the following inequality thanks to its concavity:

$$H(\alpha p_M + (1-\alpha) p_{kNN}) \ge \alpha H(p_M) + (1-\alpha) H(p_{kNN}).$$

As the entropy of adjusted distribution is lower bounded by the convex combination of entropies of prediction distribution of instance x and its nearest neighbors, we can take into account both uncertainty of them approximately.

3.2 NNS Based Test Case Selection

We number each test case in the test case set T, starting from 1 to the total number of test cases $|T|$. Each number represents a unique test case. We use some lists and matrices to save the intermediate results such as test case representations, prediction results, and distances, so that these calculation only needs to be processed once and the results can be reused, and if new test cases are added to the test case set, only new test cases need to be processed incrementally, which can improve the efficiency of test case selection. Algorithm 1 shows the basic flow of the NNS test case selection algorithm, which can be divided into the following steps:

Step 1: Representation and prediction output extraction. In this step, we extract the representation and the prediction output of each test case in the test suite. First, we use the representation extractor E to extract the representation of all test cases in the test suite to get the representation of each test case, which is saved in the representation list R in the form of vectors by their numbers (line 7). As introduced before, the representation of a test case can be extracted by the target DNN model's inner layers, or extracted by a DNN model trained by semi-supervised learning like BYOL [14], so we take the representation extractor as a parameter can be passed in. Then, we obtain the prediction output of the target DNN model M for each test case, and save the prediction output in vector form by their numbers in the prediction output list P (line 8).

Step 2: Construct a distance matrix. In this step, we construct a matrix of distances between two test cases. We compute the distance between two test case representations using the representation vectors extracted in the first step and save the results in the distance matrix DM (line 10-16). We use the cosine distance to calculate the distance between representations (see Eq. 8).

$$\begin{aligned} CosineDistance(X, Y) &= 1 - CosineSimilarity(X, Y) \\ CosineSimilarity(X, Y) &= \frac{X \cdot Y}{\|X\| \, \|Y\|} \end{aligned} \tag{8}$$

The higher the cosine similarity of the two test case representations the closer the distance is. The i-th row in the distance matrix DM is a list of the distances between test case T_i and all other test cases.

Step 3: Prediction smoothing and uncertainty calculation. In this step, we calculate the uncertainty of each test case after prediction smoothing, and use it to rank the test cases. For each test case T_i, we first get the index of the test case's k-NNs by the row DM_i in the distance matrix (line 18-29), and then take the prediction vector P_i of the test case and its prediction vector p_{kNN} of k-NN from P according to the index (line 20-23). We get the smoothed prediction p , and calculate the uncertainty of the test case using the given uncertainty measure with p as input (line 24-25). Finally, the test samples are ranked by uncertainty (line 26).

Step 4: Test case selection. In this step, test case selection is completed and returned. Based on the budget B, test cases are added to the test case subset T^S in descending order of uncertainty, and that subset T^S is returned (line 28-33).

4 EXPERIMENT DESIGN

We present our experiment design on NNS based test case selection.

4.1 Datasets and Models

For a fair comparison with TestRank [29], we refer to and follow their experimental setup of the selection of datasets and DNN models, and extend the two datasets (MNIST and Fashion-MNIST) with two DNN models (LeNet-1 and LeNet-5). In total, our experiments use five datasets, namely MNIST [28], Fashion-MNIST [49], CIFAR-10 [24], SVHN [36] and STL-10 [8]. They are commonly used datasets in the evaluation of test case selection methods.

Specifically, the MNIST dataset collected 70,000 grayscale images of 28×28 handwritten digits. The original authors splitted the data into two subsets, the training set containing 60,000 images and the test set containing 10,000 images.

The Fashion-MNIST dataset follows the MNIST dataset setting, using the same data format, the same amount of data, and the same data split, with difference that the task corresponding to this dataset is a classification task for ten categories of fashion products. These ten categories of fashion products are T-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, and ankle boot.

The CIFAR-10 dataset collects 60,000 32×32 color images of ten different types of objects. These ten types of objects are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The original authors divided the data into two subsets, the training set containing 50,000 images and the test set containing 10,000 images.

The SVHN (The Street View House Numbers) dataset collects a total of 630,420 images of 32×32 real-world house numbers, corresponding to a classification task of ten digits. The official database provides three subsets: the training set with 73,257 images, the test set with 26,032 images, and the extended set with 531,131 images.

STL-10 dataset collects 113,000 96×96 color images. These images include ten types of objects, namely, airplane, bird, car, cat, deer, dog, horse, monkey, ship, and truck. Unlike other datasets, only some of the data in the STL-10 dataset are labeled. The dataset contains a total of 13,000 labeled images, of which 5,000 are for the training set and 8,000 for the test set. The dataset also contains 100,000 unlabeled images.

According to the complexity of the task, we use the appropriate DNN model architecture for each dataset. For the MNIST and Fashion-MNIST datasets, we use LeNet-1 and LeNet-5 [27], respectively. For the CIFAR-10 dataset, we use ResNet-18 [16]. For the SVHN dataset, we use Wide-ResNet [50]. For the STL-10 dataset, we use ResNet-34 [16].

In order to simulate real-world application scenarios, the data in each dataset is re-splitted into training samples, validation samples and test samples according to different usages. The training samples are used for training the target DNN model, the validation samples are used for evaluating the accuracy of the target DNN model, and the test samples are further spiltted into labeled and unlabeled parts. The labeled test samples are only used in TestRank to train its GCN and MLP. The unlabeled test samples are used to evaluate the test case selection method. For each dataset, we train three models by sampling different distributions of data from the training samples. The IDs of the three models are A, B and C. The settings of the corresponding DNN models can be found in Table 1.

4.2 Noisy Data

Our experiments consider two types of noisy data, namely Out-of-Distribution (OOD) data and adversarial example data. Out-of-distribution data are data that are not contained in the same distribution as the data used for training the target DNN model. Broadly speaking out-of-distribution data can be either oriented to the same

Table 1: Dataset and Model Setup

Dataset	#Classes	#Training Samples	#Validation Samples	#Test Samples		DNN Architecture	Model ID	Model Acc. (%)
				Labeled	Unlabeled			
MNIST	10	20,000	1,000	7,800	41,200	LeNet-1	A	96.3
							B	97.2
							C	93.4
Fashion-MNIST	10	20,000	1,000	7,800	41,200	LeNet-5	A	88.4
							B	86.2
							C	81.5
CIFAR-10	10	20,000	1,000	7,800	31,200	ResNet-18	A	70.1
							B	66.4
							C	68.3
SVHN	10	50,000	531,000	10,000	39,000	Wide-ResNet	A	94.3
							B	92.6
							C	81.6
STL-10	10	5,000	500	1,500	6,000	ResNet-34	A	54.8
							B	54.0
							C	53.6

task as the target DNN model, or oriented to a different task. In our experiments only the former case is considered, because it is not possible to obtain correct results if inference is performed using a DNN model on data oriented to a different task, so it is not meaningful for the evaluation of test case selection methods. We selected three datasets as out-of-distribution data, including MNIST-C [35], Fashion-MNIST-C [47], CIFAR-10-C [17].

The adversarial examples are generated by the DNN adversarial attack methods, i.e., for the target DNN model, a subtle perturbation δ is added to the original data sample x, which causes the DNN model to change the prediction result for the adversarial example $x+\delta$. The perturbation to the sample is restricted to a certain range ϵ, i.e., $|\delta| < \epsilon$. We use four popular adversarial attack methods, which are FGSM [13], BIM [26], CW [6] and DeepFool [34].

4.3 Baseline Methods

We compared our approach with four uncertainty-based test case selection methods, five SA-based methods, as well as TestRank. We choose four uncertainty-based test case selection methods that are widely used and effective, which are DeepGini [10], MaxP [18, 32], Margin [42] and Entropy [5]. We choose five SA-based methods following the setup in [47], which are Distance-Based SA (DSA), Per-Class Likelihood-Based SA (PC-LSA), Per-Class Mahalanobis-Distance Based SA (PC-MDSA), Per-Class Multimodal LSA (PC-MLSA) and Per-Class Multimodal MDSA (PC-MMDSA).

We evaluated two representation extraction methods separately, using the target model itself as a representation extractor or with a dedicated representation extractor obtained by unsupervised learning. We also evaluated our methods using DeepGini, MaxP, Margin and Entropy as uncertainty calculation methods, respectively. For our method in the experiments we distinguish the uncertainty calculation methods by subscripts. The methods using the model itself as a representation extractor are labeled as: $NNS_{DeepGini}$, NNS_{MaxP}, NNS_{Margin} and $NNS_{Entropy}$. The methods using a dedicated representation extractor to extract the representation are labeled as: $NNS^{Unsup}_{DeepGini}$, NNS^{Unsup}_{MaxP}, NNS^{Unsup}_{Margin} and $NNS^{Unsup}_{Entropy}$. In our experiments the parameter k is set to 10 and the parameter α to 0.5 for NNS and NNS^{Unsup}, except where otherwise indicated.

For the MNIST dataset, we use the Basic Autoencoder [19] as a dedicated representation extractor. For the Fashion-MNIST dataset, we use the Convolutional AutoEncoder [33] as a dedicated representation extractor. For the CIFAR-10, SVHN and STL-10 datasets, we use BYOL [14] as a dedicated representation extractor.

4.4 Evaluation Metrics

We use Average Percentage of Fault Detection (APFD) [41] to evaluate the overall performance of a test case selection method. We also use Test Relative Coverage (TRC) [29] to evaluate the performance of a test case selection method for a given budget.

APFD is widely used to evaluate the overall performance of a test case selection method, and is calculated by Eq. 9.

$$APFD = 1 - \frac{TF_1 + TF_2 + \cdots + TF_m}{nm} + \frac{1}{2n} \tag{9}$$

where n denotes the total number of test cases in test suite, m denotes is the total number of faults exposed in the software under test suite, and TF_i denotes the position of the first test in test suite that exposes fault i. The value of APFD ranges from 0 to 1, with higher values representing higher fault detection efficiency.

APFD is a good measure of the overall performance of test case selection methods, but is not suitable for comparing different test case selection methods for a given budget. Therefore, we also use TRC to compare the performance of different test effort selection methods for a given budget. TRC can be expressed by Eq. 10.

$$TRC = \frac{\#DetectedFailures}{\min(\#Budget, \#TotalFailures)} \tag{10}$$

TRC measures how far a test input selection method is to the ideal case in a given budget. the value of TRC also ranges from 0 to 1, the higher the value represents the higher the proportion of faults detected in a given budget.

4.5 Research Questions

We attempt to answer the following three research questions (RQs):

RQ1: Effectiveness on Clean Data. How effective is our test case selection method on clean data?

We first want to know how well our method works on a test dataset with the same distribution as the training data of the DNN,

Table 2: APFD on Clean Data

	Dataset	MNIST			Fashion-MNIST			CIFAR-10			SVHN			STL-10		
	Model ID	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C
	Ideal	98.78	98.61	96.76	93.67	93.55	90.43	84.92	82.91	82.87	94.93	94.08	88.30	80.37	78.69	76.99
	TestRank	90.06	90.18	93.17	83.17	**83.38**	**83.88**	74.66	73.67	74.30	85.63	87.59	84.76	70.20	69.13	**67.71**
SA	DSA	90.81	90.53	90.10	77.94	77.51	76.52	67.22	64.84	66.06	85.08	84.44	79.46	64.67	61.84	61.26
	PC-LSA	89.46	80.27	48.59	70.48	66.89	67.17	66.31	60.10	64.02	56.67	50.98	45.00	53.60	56.81	55.75
	PC-MDSA	89.89	81.53	93.10	68.05	62.88	63.09	68.25	61.89	65.75	76.28	67.20	60.84	60.63	60.81	59.45
	PC-MLSA	89.67	76.17	87.40	69.83	59.54	65.47	57.38	46.79	51.14	66.43	55.61	59.48	62.68	59.67	60.10
	PC-MMDSA	78.25	76.67	85.50	61.57	58.73	58.27	56.56	58.44	58.22	71.33	70.42	56.50	46.38	52.89	47.23
Uncertainty	DeepGini	94.89	94.01	90.93	83.36	82.58	79.81	70.97	67.90	69.74	85.90	84.85	82.00	66.01	64.13	62.55
	MaxP	94.88	94.00	90.93	83.34	82.54	79.75	70.92	67.86	69.70	85.88	84.82	81.88	65.95	64.05	62.47
	Margin	94.86	93.97	90.89	83.21	82.39	79.49	70.76	67.72	69.57	85.78	84.72	81.54	65.81	63.79	62.28
	Entropy	94.90	94.00	90.96	83.36	82.60	80.08	71.12	68.05	69.88	85.95	84.89	82.30	66.15	64.35	62.74
NNS	DeepGini	96.48	96.02	93.68	84.09	83.00	80.34	72.42	70.02	70.55	89.95	89.86	85.25	66.51	64.82	62.36
	MaxP	**96.52**	**96.06**	**93.68**	84.09	82.97	80.30	72.33	69.95	70.47	**90.05**	90.00	85.15	66.40	64.65	62.20
	Margin	96.48	96.03	93.57	83.93	82.79	80.09	71.98	69.63	70.20	90.02	**90.00**	84.76	66.11	64.15	61.84
	Entropy	96.33	95.86	93.61	83.89	82.88	80.56	72.50	70.03	70.69	89.70	89.52	**85.41**	66.67	65.14	62.63
NNS^{Unsup}	DeepGini	95.68	95.66	93.25	84.11	83.01	80.02	75.99	73.79	74.53	86.39	86.98	81.79	72.83	71.19	67.03
	MaxP	95.76	95.72	93.25	**84.12**	82.98	80.00	**76.91**	74.66	**74.99**	87.07	88.05	81.83	**73.64**	**71.67**	67.45
	Margin	95.70	95.63	93.09	83.85	82.72	79.76	76.82	**74.66**	74.57	86.88	88.18	81.16	73.62	71.38	67.27
	Entropy	95.43	95.42	93.17	83.80	82.77	80.24	73.84	71.69	73.28	84.82	84.81	81.62	71.20	70.00	66.17

and whether it can detect more faults faster. We use the test case selection method mentioned in Section 4.3 on the unlabeled test samples from the original dataset mentioned in Section 4.1. We used 1%, 2%, 5%, 10%, 15%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90% and 100% of the unlabeled test samples as the budget for test case selection, respectively. We conducted experiments on each dataset separately to calculate the APFD and TRC scores for each test case selection method as a way to compare various methods.

RQ2: Effectiveness on Noisy Data. How effective is our test case selection method on OOD data and adversarial example data?

In the real world, the data processed by the deep learning system in the operational environment often differ in distribution from the data used for training, such as different lighting and contrast of the images. To verify the effectiveness of the method on data with different distribution from the training data, i.e., noisy data, we use two types of noisy data, OOD data and adversarial example data as mentioned in Section 4.2 . For each dataset, we randomly select the same number of OOD data from the corresponding OOD data as the unlabeled test data to evaluate the effectiveness of the method on the OOD data. Also the same number of adversarial examples as the unlabeled test data were randomly selected from the corresponding adversarial examples to evaluate the effectiveness of the method on the adversarial example data. We used the same test case selection budget as in **RQ1**. Consistent with **RQ1**, we also calculated the APFD and TRC scores for each method separately.

RQ3: Efficiency. Is our method more efficient than TestRank and SA-based test case selection methods?

We compared the efficiency of the methods by the time to complete a test case selection process in a test suite.

RQ4: Impact of Parameters. How does different values for the number of nearest neighbors k and the smoothing weight parameter α NNS affect the effectiveness of test case selection?

The NNS method contains two parameters the number of nearest neighbors k and the smoothing weight α. Different values for the parameters may affect the effect of NNS. To give guidance on the values of k and α, we did ablation experiments on them. First, we fixed the parameter α to 0.5 and changed k to 0, 1, 2, 3, 5, 7, 10, 20, 50, 70, and 100 to obtain the APFD scores for various cases. And then, we fixed the parameter k to 10 and changed α to 0, 0.1, 0.2, . . . , 0.9, 1.0, and obtained the APFD scores for various values.

5 EXPERIMENT RESULTS

We present the results of our experiments, and we also summarize and analyze the results. We implement our approach in Python with Pytorch V1.9.0. The experiments were performed on Ubuntu 20.04 with an Intel i9 CPU, a NVIDIA GeForce RTX 3080 GPU and 32GB DDR4 RAM. For the implementation of TestRank, we directly use its official code[1]. For the implementation of SA-based methods, we invoke the API provided by the DNN-TIP library[2].

5.1 Effectiveness on Clean Data (RQ1)

On clean data we evaluated two groups of NNS methods, NNS using a target DNN model to extract representations and NNS^{Unsup} using a dedicated representation extractor, and compared them with a group of uncertainty-based test case selection methods and also with TestRank.

Table 2 shows the comparison of the APFD obtained by each test selection method on clean data, in which the items in bold indicate the highest score of APFD. The third row of the table shows the best APFD score that can be obtained in the ideal case, for the upper limit of APFD for the test case selection method for that dataset and the target DNN model.

Figure 2 shows the mean values of the TRC scores obtained for each method under different budgets. The horizontal axis is the budget for test case selection, the values are as a percentage of the total sample, and the vertical axis is the TRC scores. When evaluating test case selection methods using TRC scores, we are

[1]https://github.com/cure-lab/TestRank
[2]https://github.com/testingautomated-usi/dnn-tip

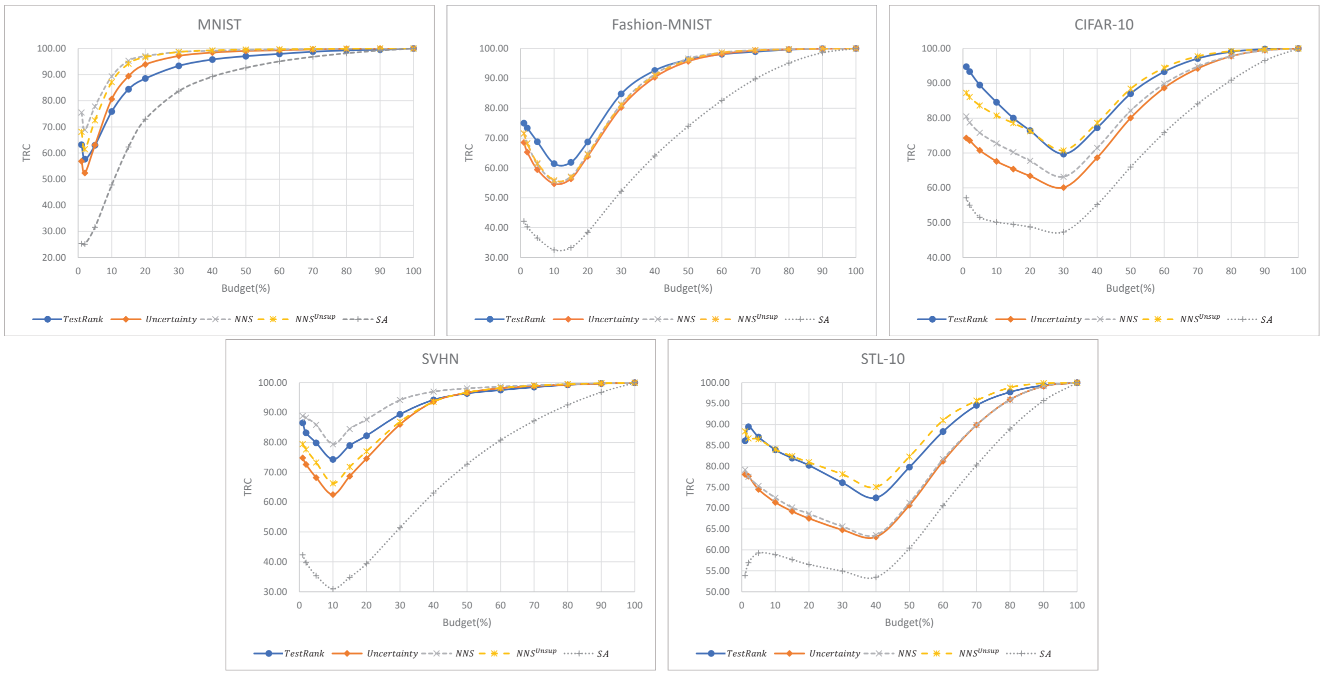

Figure 2: TRC on Clean Data

more concerned with smaller budgets because when the budget is large enough to cover the entire set of test cases, all test cases are selected and the TRC for all test case selection methods must be 100%, whereas a higher TRC score when the budget is small indicates that more problems can be revealed at a smaller cost. As mentioned in [29], the TRC score rises and then falls as the budget increases, with the turning point being the point at which the budget equals the total number of faults in the test case set. The difference between the TRC scores obtained by the various methods is more stable before the turning point, and gradually decreases after the turning point.

Comparing with uncertainty-based methods. As show in Table 2, NNS and NNS^{Unsup} achieve the highest APFD scores in all cases when $TestRank$ is excluded. On MNIST dataset, NNS and NNS^{Unsup} achieve 2.09% and 1.54% higher average APFD score than $Uncertainty$. On Fashion-MNIST dataset, NNS and NNS^{Unsup} achieve 0.73% and 0.6% higher average APFD score than $Uncertainty$. On CIFAR-10 dataset, NNS and NNS^{Unsup} achieve 1.38% and 5.13% higher average APFD score than $Uncertainty$. On SVHN dataset, NNS and NNS^{Unsup} achieve 4.10% and 0.76% higher average APFD score than $Uncertainty$. On STL-10 dataset, NNS and NNS^{Unsup} achieve 0.27% and 6.1% higher average APFD score than $Uncertainty$, respectively. It can be seen that our method can enhance the uncertainty-based test case selection method, but the degree of improvement is different by the different representation extraction methods. This suggests that the choice of data representation has a direct impact on the effectiveness of test case selection.

It can be seen from Figure 2 that the average TRC scores of NNS and NNS^{Unsup} are all higher than $Uncertainty$ before the turning point in each plot. For quantitative comparison, we compare the average TRC scores before the turning point for each method. NNS achieves 32.16% higher than $Uncertainty$ on the MNIST dataset, NNS^{Unsup} achieves 4.21% higher than $Uncertainty$ on the Fashion-MNIST dataset, NNS^{Unsup} achieves 18.59% higher than $Uncertainty$ on the CIFAR-10 dataset, and NNS achieves 23.5% higher than $Uncertainty$ on the SVHN dataset, and NNS^{Unsup} achieves 17.14% higher than $Uncertainty$ on the STL-10 dataset. This shows a significant improvement of our method compared to $Uncertainty$.

Comparing with SA-based methods. It can be seen from Table 2 that DSA can achieve higher APFD scores than other SA-based methods in most cases. The APFD scores achieved by NNS and NNS^{Unsup} on each data set are higher than those of DSA. On MNIST dataset, NNS and NNS^{Unsup} achieve 5.39% and 4.79% higher average APFD score than DSA. On Fashion-MNIST dataset, NNS and NNS^{Unsup} achieve 6.58% and 6.41% higher average APFD score than DSA. On CIFAR-10 dataset, NNS and NNS^{Unsup} achieve 7.35% and 13.03% higher average APFD score than DSA. On SVHN dataset, NNS and NNS^{Unsup} achieve 6.40% and 2.38% higher average APFD score than DSA. On STL-10 dataset, NNS and NNS^{Unsup} achieve 2.98% and 12.30% higher average APFD score than DSA. It can be seen from Figure 2 that the TRC scores achieved by the SA-based methods also have a significant disadvantage compared to other methods.

Comparing with TestRank. From the Table 2, on a total of 15 models across the five datasets, our methods NNS and NNS^{Unsup} get the highest APFD scores in both six cases, for a total of 12 cases accounting for 80%, while $TestRank$ gets the highest APFD scores in three cases accounting for 20%. It can be seen from Figure 2 that on the MNIST and SVHN datasets, the TRC scores of the NNS method are significantly higher than $TestRank$ before the turning point, and on the CIFAR-10 and STL-10 datasets, NNS^{Unsup} is comparable to $TestRank$. Only on the Fashion-MNIST dataset, $TestRank$ outperform our method. This may be because that the

Table 3: APFD on OOD Data

	Dataset	MNIST			Fashion-MNIST			CIFAR-10		
	Model ID	A	B	C	A	B	C	A	B	C
	Ideal	91.42	90.24	91.03	71.60	72.48	70.49	79.69	77.92	78.93
	TestRank	75.70	77.21	73.81	56.57	59.43	56.81	66.60	66.40	66.90
SA	DSA	79.50	78.79	77.01	58.51	60.08	57.74	62.70	60.77	61.79
	PC-LSA	68.68	66.69	49.66	**59.51**	58.52	58.11	60.87	57.11	59.34
	PC-MDSA	65.57	65.09	70.72	57.74	57.23	56.74	62.38	58.70	61.27
	PC-MLSA	66.80	63.77	67.81	58.50	56.59	57.08	54.89	45.64	54.19
	PC-MMDSA	67.66	64.36	68.42	59.12	**60.12**	**58.51**	53.97	54.61	55.46
$Uncertainty$	DeepGini	78.41	78.94	77.67	57.86	59.48	56.40	65.24	63.28	65.29
	MaxP	78.55	78.86	77.72	57.79	59.37	56.34	65.18	63.22	65.24
	Margin	78.62	78.71	77.77	57.60	59.12	56.11	65.03	63.06	65.09
	Entropy	78.04	79.06	77.45	58.03	59.67	56.58	65.39	63.46	65.45
NNS	DeepGini	83.90	83.27	84.28	57.60	59.66	56.17	66.77	64.98	66.03
	MaxP	84.34	**83.31**	**84.50**	57.53	59.53	56.10	66.64	64.84	65.93
	Margin	**84.50**	83.02	84.47	57.30	59.18	55.83	66.28	64.43	65.65
	Entropy	82.95	83.11	83.61	57.78	59.89	56.35	66.87	65.08	66.11
NNS^{Unsup}	DeepGini	79.08	77.56	78.61	55.96	58.18	54.62	68.51	66.52	68.25
	MaxP	79.82	77.83	79.28	55.89	58.02	54.55	**69.25**	**67.05**	**68.81**
	Margin	80.03	77.49	79.29	55.61	57.55	54.23	69.11	66.80	68.52
	Entropy	77.93	77.19	77.51	56.14	58.48	54.82	66.49	64.88	66.76

data in Fashion-MNIST dataset are all small-sized grayscale images, which are relatively concentrated in representation space, and $TestRank$ is able to capture finer patterns through learning with GCN and MLP.

Comparing with ideal. We can see from the Table 2 that the APFD scores of our methods are close to the ideal situation. The best case is NNS_{MaxP} with model A on MNIST dataset, which is 2.25% lower than ideal. And the worst case is NNS_{Margin} with model C on STL-10 dataset, which is 15.15% lower than ideal. On average, our methods is 8.22% lower than ideal.

Answer to RQ1. On clean data, our method can consistently significantly improve the performance of uncertainty-based test case selection methods. Our method is quite competitive with TestRank, and significantly outperforms SA-based methods.

5.2 Effectiveness on Noisy Data (RQ2)

We evaluated the OOD data and the adversarial example data separately as noisy data using our method. Table 3 shows a comparison of the APFDs obtained for each test selection method on the OOD data, and Table 4 shows the comparison of APFDs obtained for each test selection method on the adversarial example data.

Comparing with uncertainty-based methods. On OOD data, NNS and NNS^{Unsup} get the highest APFD scores in 7 out of 9 cases when $TestRank$ is excluded. On average, NNS and NNS^{Unsup} achieve 2.19% and 0.49% higher APFD than $Uncertainty$. On adversarial examples, NNS and NNS^{Unsup} get the highest APFD scores in 14 out of 15 cases when $TestRank$ is excluded. On average, NNS achieve 3.26% higher APFD than $Uncertainty$, but NNS^{Unsup} achieve 0.45% lower than $Uncertainty$. The results show that our method can improve the performance of uncertainty-based test case selection methods on noisy data. For adversarial examples, using self extracted representation seems more effective.

Comparing with SA-based methods. On OOD data, NNS and NNS^{Unsup} get the highest APFD scores in 6 out of 9 cases, and SA-based methods get the highest APFD scores in 3 cases. On adversarial examples, NNS and NNS^{Unsup} get the highest APFD scores in 5 out of 15 cases, and SA-based methods get the highest APFD scores in 10 cases. The results show that the SA-based methods have a significant advantage over the other methods on adversarial examples.

Comparing with TestRank. On average, NNS and NNS^{Unsup} achieve 2.50% and 0.80% higher APFD than $TestRank$. On average, NNS achieve 3.26% higher APFD than $Uncertainty$, but NNS^{Unsup} achieve 0.45% lower than $TestRank$. Our methods are more effective on OOD and adversarial example data compared to $TestRank$.

Comparing with ideal. We find that all the methods we evaluated have greater gap in APFD on noisy data than on clean data. On OOD data, NNS and NNS^{Unsup} achieve 11.32% and 13.02% lower average APFD than ideal situation. On adversarial examples, NNS and NNS^{Unsup} achieve 11.50% and 15.20% lower average APFD than ideal situation. That means noisy data is a harder task for test case selection.

Answer to RQ2. Our method are more effective on OOD and adversarial example data compared to $TestRank$ and uncertainty-based methods. For adversarial examples, using self extracted representation are more effective. SA-based methods have a significant advantage over the other methods on adversarial examples.

5.3 Efficiency (RQ3)

Table 5 shows the time cost of each test case selection methods. Due to space limitation, we only show the results of $NNS_{DeepGini}$, $NNS^{Unsup}_{DeepGini}$, $TestRank$ and SA-based methods, and the results of other NNS variants are similar. We can see from the table that the time cost of TestRank and SA-based methods are more than an order of magnitude higher than our method. The reasons for this gap will be discussed in detail in Section 5.5.

Answer to RQ3. Our method is much more efficient than TestRank and SA-based test case selection methods.

5.4 Impact of Parameters (RQ4)

Since we wanted to obtain the impact of different values of the parameters on the overall performance of the test case selection method, we study the impact of the parameters by looking at the change in APFD scores. Figure 3 shows the effect of the number of nearest neighbours parameter k, i.e. the effect of taking different values of k on the APFD score when α is fixed to 0.5. The horizontal axis is the values of k and the vertical axis is the values of APFD. The three lines in the figure represent each of the three models for the dataset. We observe that when the value of k is small, the APFD score increases rapidly with increasing values of k. It can be seen that considering the nearest neighbours is effective. On the MNIST and Fashion-MNIST datasets, the APFD score decreases slowly after reaching its maximum value as the value of k increases. On the CIFAR-10, SVHN and STL-10 datasets, the APFD scores remain relatively stable when the value of k is large. Considering too many nearest neighbours is unnecessary, taking too large of k not only introduces more computational overhead, but may also bring about a loss in performance. We observe that the highest APFD scores on different datasets generally can be achieved when $k \in [5, 20]$, so in general a value of 10 for k is a appropriate choice.

Figure 4 shows the impact of the smoothing weight parameter α, i.e. the effect of taking different values of α on the APFD score when k is fixed at 10. The horizontal axis is the values of α and the vertical axis is the APFD values. When $\alpha = 0$, this means that the target DNN model's predictions for k-NN are used exclusively, regardless of the target DNN model's predictions for the test case

Table 4: APFD on Adversarial Data

	Dataset	MNIST			Fashion-MNIST			CIFAR-10			SVHN			STL-10		
	Model ID	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C
	Ideal	86.49	88.36	87.61	75.36	80.57	78.79	83.29	82.87	82.66	89.25	88.76	83.92	81.93	82.01	81.08
	TestRank	70.17	77.64	79.63	68.78	70.81	72.04	56.65	63.60	57.28	76.67	79.67	77.82	58.69	61.45	60.62
SA	DSA	**83.78**	**84.68**	**82.50**	**71.09**	**75.15**	**73.07**	65.49	65.00	66.47	78.26	81.90	72.87	66.51	66.16	66.04
	PC-LSA	83.35	76.12	49.98	69.43	72.67	71.75	**69.19**	64.54	67.80	71.25	59.66	47.49	63.50	63.58	65.75
	PC-MDSA	82.46	75.31	80.49	67.18	69.88	69.20	69.10	65.18	68.10	**81.44**	76.67	66.08	66.52	68.28	67.75
	PC-MLSA	82.83	73.43	78.33	68.31	66.95	70.45	65.57	51.92	57.53	74.82	65.76	64.49	68.03	67.41	**68.41**
	PC-MMDSA	78.84	76.55	75.75	64.14	69.63	67.72	59.84	61.48	60.71	79.60	78.07	63.67	**68.80**	57.97	58.69
Uncertainty	DeepGini	62.06	74.31	76.02	61.89	63.43	63.81	67.20	68.02	69.46	66.60	80.01	76.37	66.90	68.35	67.59
	MaxP	62.02	74.29	76.00	61.88	63.30	63.66	67.17	67.98	69.43	66.59	79.98	76.28	66.86	68.32	67.52
	Margin	61.81	74.23	75.93	61.56	62.97	63.24	67.08	67.85	69.32	66.57	79.86	76.02	66.72	68.12	67.35
	Entropy	62.20	74.41	76.13	62.05	63.76	64.29	67.29	68.16	69.59	66.58	80.10	76.58	67.02	68.52	67.76
NNS	DeepGini	75.79	81.71	81.45	65.29	66.84	65.86	68.39	**69.11**	69.53	70.46	84.58	**79.29**	67.32	68.55	67.31
	MaxP	75.74	81.63	81.40	65.22	66.63	65.66	68.34	69.06	69.46	70.47	**84.65**	79.28	67.26	68.46	67.19
	Margin	75.52	81.43	81.20	64.79	66.10	65.12	68.10	68.80	69.23	70.32	84.53	79.12	67.01	68.10	66.91
	Entropy	75.91	81.86	81.55	65.41	67.31	66.41	68.46	69.10	**69.61**	70.48	84.44	79.27	67.50	**68.83**	67.54
NNS^{Unsup}	DeepGini	78.91	82.00	80.94	65.64	67.25	66.53	68.15	67.09	67.04	45.48	74.72	68.91	66.20	66.14	62.57
	MaxP	78.73	81.82	80.74	65.67	67.06	66.34	68.55	67.73	67.42	46.19	76.25	70.18	66.38	66.30	62.88
	Margin	78.25	81.34	80.22	65.32	66.50	65.72	67.70	67.22	66.71	46.15	77.57	71.66	65.35	65.19	62.17
	Entropy	79.21	82.35	81.30	65.64	67.74	67.15	66.52	65.09	65.82	44.66	69.91	66.74	64.53	64.61	61.40

Table 5: Test Case Selection Time Cost (s)

	$NNS_{DeepGini}$	$NNS^{Unsup}_{DeepGini}$	TestRank	DSA	PC-LSA	PC-MDSA	PC-MLSA	PC-MMDSA
MNIST	17.57	6.70	759.12	1892.32	271.35	107.11	191.02	239.22
Fashion-MNIST	9.87	9.83	875.17	1850.15	270.58	106.86	170.02	220.21
CIFAR-10	26.65	26.55	642.35	1928.72	227.80	215.99	180.55	322.30
SVHN	49.02	49.70	667.67	2004.35	285.42	113.45	202.33	251.38
STL-10	2.59	2.37	110.40	259.03	37.14	15.66	26.15	32.75

itself. As can be seen from the figure this is not the best choice. When $\alpha = 1$, it degrades to the original uncertainty-based test case selection method. We observe that the highest APFD scores on different datasets generally occur in the middle range, so in general a value of 0.5 is an appropriate choice.

Answer to RQ4. Our method can be significantly improved by considering only a small number of nearest neighbors. Considering too many nearest neighbors may degrade performance. In general, a value of 10 for k is a good choice. For the other hand, using only the uncertainty of the test case itself or only the uncertainty of the k nearest neighbor, can not achieve the best results. In general, a value of 0.5 for α is a good choice.

5.5 Discussion

Although our method does not have a particularly significant advantage over TestRank in terms of effectiveness, as demonstrated in our experiments. Compared to TestRank, a learning-based test case selection method, our method is lightweight and easy to use. Specifically, the main advantages of our approach over TestRank are as follows:

- No additional training overhead is required. TestRank requires a GCN and an MLP to be trained before it can be used. 450 epochs are needed to train the GCN and 150 epochs to train the MLP in its official implementation. Our approach avoids this part of the overhead.
- No additional labeling effort required. TestRank requires a separate labeled dataset to train the ranking model, which introduces additional labeling effort. And our method no labeled data is needed, only unlabeled data.
- Smaller overhead for calculating k-NN. When training GCN, TestRank needs to use representation to build a topology graph between the test case and its k-NN, and the recommended k value is 100. Our experimental results show that when $k = 10$, our method can get a good performance, and our method does not require the construction of a topology graph.
- Good flexibility. TestRank needs to retrain the ranking model when the test suite changes significantly. And our method can be used directly on the new test suite.

There is one point worth noting in the experimental results, the SA-based test case selection method shows some advantages on the adversarial examples. We think this is because the SA of a test sample reflects the novelty of the sample compared to the training data, which can measure whether the test sample is in the distribution of the training data, and it can be interpreted as an OOD data detection metric. This also explains why the SA-based method is less effective than other methods on clean data with the same distribution as the training data.

From our experimental results, we can see that there are significant differences in the performance of NNS and NNS^{Unsup} under different circumstances, which means the choice of representation matters. Comparing the results of RQ1 and RQ2, we notice that NNS works better on clean and OOD data for MNIST dataset, but NNS^{Unsup} works better on adversarial examples. Similar phenomenon can be observed on other datasets. We deduce that NNS works

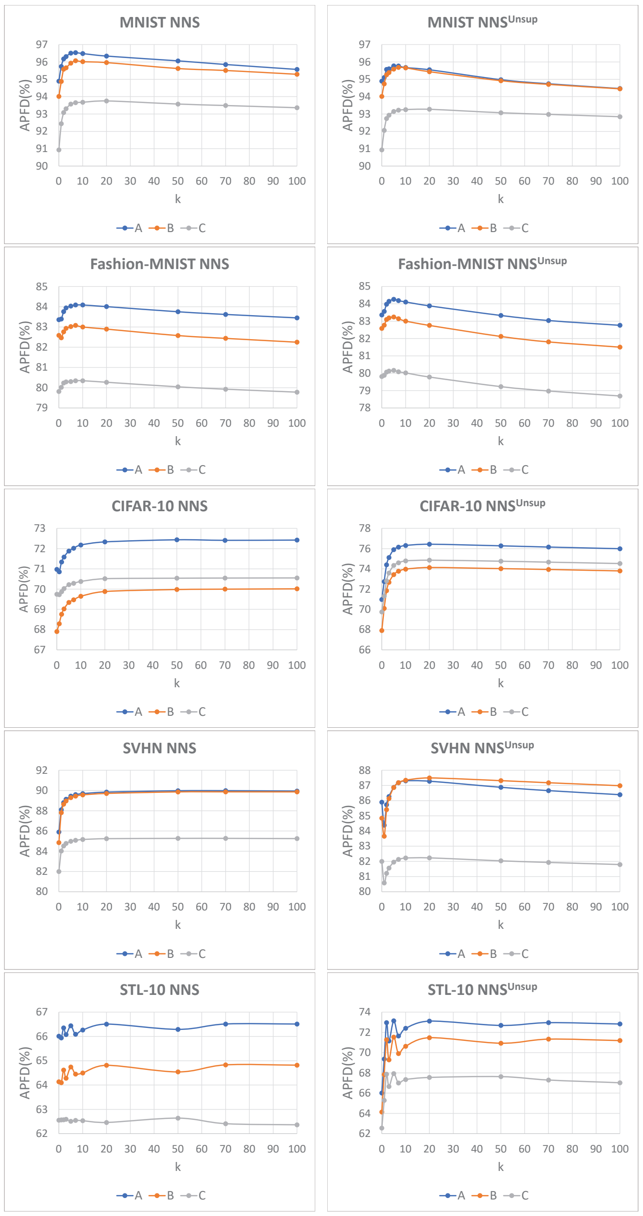

Figure 3: The Impact of k

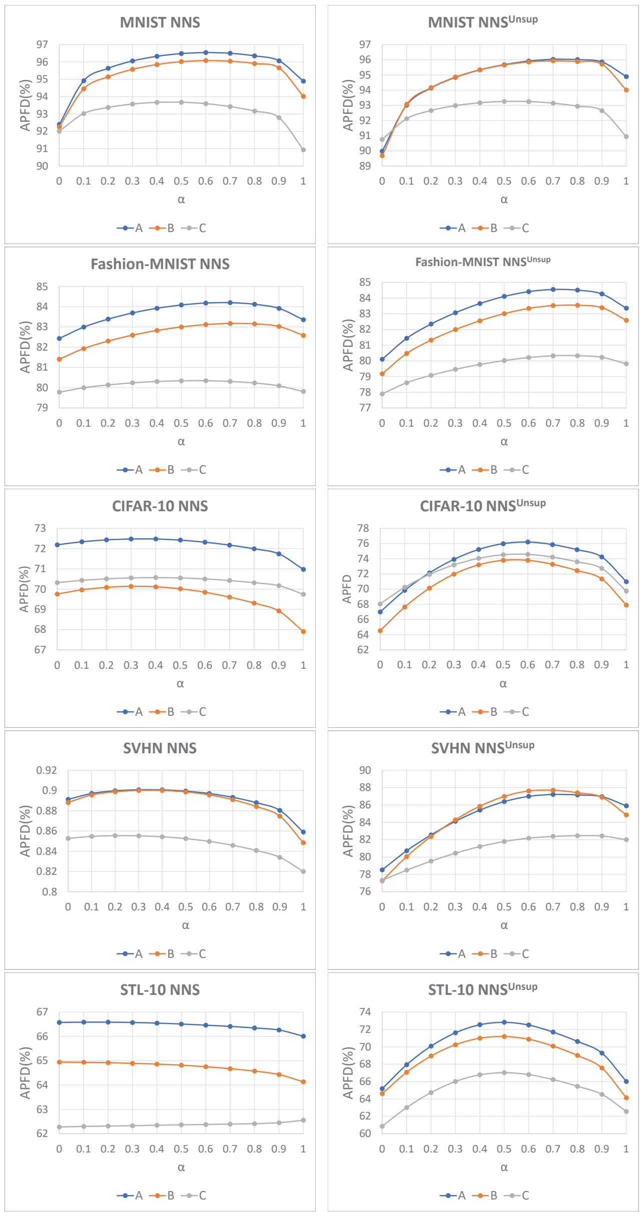

Figure 4: The Impact of α

well when the DNN model is well trained on clean data, such as on MNIST and on SVHN. And adversarial examples are used to mislead the DNN model on purposes, so it will have a greater impact on the method that are better on clean data. We leave it as our future work to investigate the guidance on how to select representations on different types of data.

6 CONCLUSIONS

In this paper, we propose a novel and lightweight DNN test selection method, namely NNS, to improve testing efficiency and reduce labeling cost. The key design idea is to calibrate the uncertainty by utilizing k-nearest neighbors prediction smoothing to give the error-prone test cases higher priority. Our method is applied to four uncertainty-based test case selection methods with two representation extraction variations. The results of the experiments demonstrate that our method can consistently improve the performance of uncertainty-based test case selection methods, and outperform TestRank which is a learning based test case selection method. Our method is a plug-and-play technique that can be easily integrated with different uncertainty measures and different representation learning methods.

ACKNOWLEDGMENTS

This work was supported by the National Key R&D Program of China (2021ZD0112903).

REFERENCES

[1] Dara Bahri and Heinrich Jiang. 2021. Locally Adaptive Label Smoothing Improves Predictive Churn. In *Proceedings of the 38th International Conference on Machine Learning*. 532–542.

[2] Dara Bahri, Heinrich Jiang, and Maya R. Gupta. 2020. Deep k-NN for Noisy Labels. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 540–550.

[3] Robert J. N. Baldock, Hartmut Maennel, and Behnam Neyshabur. 2021. Deep Learning Through the Lens of Example Difficulty. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (Eds.). 10876–10889.

[4] Jacob Bogage. 2016. Tesla driver using autopilot killed in crash. https://www.washingtonpost.com/news/the-switch/wp/2016/06/30/tesla-owner-killed-in-fatal-crash-while-car-was-on-autopilot/

[5] Taejoon Byun, Vaibhav Sharma, Abhishek Vijayakumar, Sanjai Rayadurgam, and Darren D. Cofer. 2019. Input Prioritization for Testing Neural Networks. In *Proceedings of the IEEE International Conference On Artificial Intelligence Testing*. 63–70.

[6] Nicholas Carlini and David A. Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 39–57.

[7] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. 2020. A Simple Framework for Contrastive Learning of Visual Representations. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 1597–1607.

[8] Adam Coates, Andrew Y. Ng, and Honglak Lee. 2011. An Analysis of Single-Layer Networks in Unsupervised Feature Learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011 (JMLR Proceedings, Vol. 15)*, Geoffrey J. Gordon, David B. Dunson, and Miroslav Dudík (Eds.). JMLR.org, 215–223.

[9] Thomas M Cover. 1968. Rates of convergence for nearest neighbor procedures. In *Proceedings of the Hawaii International Conference on Systems Sciences*, Vol. 415.

[10] 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 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 177–188.

[11] Evelyn Fix and Joseph L. Hodges. 1989. Discriminatory Analysis - Nonparametric Discrimination: Consistency Properties. *International Statistical Review* 57 (1989), 238.

[12] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. 2016. *Deep Learning*. MIT Press.

[13] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.).

[14] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Ávila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. 2020. Bootstrap Your Own Latent - A New Approach to Self-Supervised Learning. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.).

[15] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2017. On Calibration of Modern Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017 (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). PMLR, 1321–1330.

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

[17] Dan Hendrycks and Thomas G. Dietterich. 2019. Benchmarking Neural Network Robustness to Common Corruptions and Perturbations. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

[18] Dan Hendrycks and Kevin Gimpel. 2017. A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks. In *Proceedings of the 5th International Conference on Learning Representations*.

[19] Geoffrey E Hinton and Ruslan R Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *science* 313, 5786 (2006), 504–507.

[20] Qiang Hu, Yuejun Guo, Maxime Cordy, Xiaofei Xie, Wei Ma, Mike Papadakis, and Yves Le Traon. 2021. Towards Exploring the Limitations of Active Learning: An Empirical Study. In *36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021*. IEEE, 917–929.

[21] Heinrich Jiang, Been Kim, Melody Y. Guan, and Maya R. Gupta. 2018. To Trust Or Not To Trust A Classifier. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (Eds.). 5546–5557.

[22] Jinhan Kim, Robert Feldt, and Shin Yoo. 2019. Guiding Deep Learning System Testing Using Surprise Adequacy. In *Proceedings of the 41st International Conference on Software Engineering*. 1039–1049.

[23] Jinhan Kim, Jeongil Ju, Robert Feldt, and Shin Yoo. 2020. Reducing DNN labelling cost using surprise adequacy: an industrial case study for autonomous driving. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann (Eds.). ACM, 1466–1476.

[24] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. 2009. *The CIFAR-10 dataset*. https://www.cs.toronto.edu/%7ekriz/cifar.html

[25] Volodymyr Kuleshov and Percy Liang. 2015. Calibrated Structured Prediction. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett (Eds.). 3474–3482.

[26] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. 2017. Adversarial examples in the physical world. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net.

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

[28] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. 1998. *The MNIST database of handwritten digits*. http://yann.lecun.com/exdb/mnist/

[29] Yu Li, Min Li, Qiuxia Lai, Yannan Liu, and Qiang Xu. 2021. TestRank: Bringing Order into Unlabeled Test Instances for Deep Learning Tasks. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (Eds.). 20874–20886.

[30] Lei Ma, Felix Juefei-Xu, Minhui Xue, Bo Li, Li Li, Yang Liu, and Jianjun Zhao. 2019. DeepCT: Tomographic Combinatorial Testing for Deep Learning Systems. In *26th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2019, Hangzhou, China, February 24-27, 2019*, Xinyu Wang, David Lo, and Emad Shihab (Eds.). IEEE, 614–618.

[31] 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 33rd ACM/IEEE International Conference on Automated Software Engineering*. 120–131.

[32] 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.

[33] Jonathan Masci, Ueli Meier, Dan C. Ciresan, and Jürgen Schmidhuber. 2011. Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction. In *Artificial Neural Networks and Machine Learning - ICANN 2011 - 21st International Conference on Artificial Neural Networks, Espoo, Finland, June 14-17, 2011, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 6791)*, Timo Honkela, Wlodzislaw Duch, Mark A. Girolami, and Samuel Kaski (Eds.). Springer, 52–59.

[34] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. 2016. DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2574–2582.

[35] Norman Mu and Justin Gilmer. 2019. MNIST-C: A Robustness Benchmark for Computer Vision. *ArXiv* abs/1906.02337 (2019).

[36] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. 2011. *Reading digits in natural images with unsupervised feature learning*. http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf

[37] Anh Mai Nguyen, Jason Yosinski, and Jeff Clune. 2015. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, 427–436.

[38] Nicolas Papernot and Patrick Mcdaniel. 2018. Deep k-Nearest Neighbors: Towards Confident, Interpretable and Robust Deep Learning. *ArXiv* abs/1803.04765 (2018).

[39] 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.

[40] Foster J. Provost, Tom Fawcett, and Ron Kohavi. 1998. The Case against Accuracy Estimation for Comparing Induction Algorithms. In *Proceedings of the Fifteenth*

International Conference on Machine Learning (ICML 1998), Madison, Wisconsin, USA, July 24-27, 1998, Jude W. Shavlik (Ed.). Morgan Kaufmann, 445–453.

[41] Gregg Rothermel, Roland H. Untch, Chengyun Chu, and Mary Jean Harrold. 2001. Prioritizing test cases for regression testing. *IEEE Transactions on software engineering* 27, 10 (2001), 929–948.

[42] Tobias Scheffer, Christian Decomain, and Stefan Wrobel. 2001. Active Hidden Markov Models for Information Extraction. In *Proceedings of the 4th International Conference on Advances in Intelligent Data Analysis*. 309–318.

[43] Weijun Shen, Yanhui Li, Lin Chen, Yuanlei Han, Yuming Zhou, and Baowen Xu. 2020. Multiple-Boundary Clustering and Prioritization to Promote Neural Network Retraining. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. 410–422.

[44] 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, Montpellier, France, September 3-7, 2018*, Marianne Huchard, Christian Kästner, and Gordon Fraser (Eds.). ACM, 109–119.

[45] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the Inception Architecture for Computer Vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2818–2826.

[46] Michael Weiss, Rwiddhi Chakraborty, and Paolo Tonella. 2021. A Review and Refinement of Surprise Adequacy. In *3rd IEEE/ACM International Workshop on Deep Learning for Testing and Testing for Deep Learning, DeepTest@ICSE 2021, Madrid, Spain, June 1, 2021*. IEEE, 17–24.

[47] Michael Weiss and Paolo Tonella. 2022. Simple techniques work surprisingly well for neural network test prioritization and active learning (replicability study). In *ISSTA '22: 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, South Korea, July 18 - 22, 2022*, Sukyoung Ryu and Yannis Smaragdakis (Eds.). ACM, 139–150.

[48] Dennis L. Wilson. 1972. Asymptotic Properties of Nearest Neighbor Rules Using Edited Data. *IEEE Trans. Syst. Man Cybern.* 2, 3 (1972), 408–421.

[49] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv preprint arXiv:1708.07747* (2017).

[50] Sergey Zagoruyko and Nikos Komodakis. 2016. Wide Residual Networks. In *Proceedings of the British Machine Vision Conference 2016, BMVC 2016, York, UK, September 19-22, 2016*, Richard C. Wilson, Edwin R. Hancock, and William A. P. Smith (Eds.). BMVA Press.

Received 2023-02-16; accepted 2023-05-03