Practical Accuracy Estimation for Efficient Deep Neural Network Testing

JUNJIE CHEN, College of Intelligence and Computing, Tianjin University, China
ZHUO WU, Tianjin International Engineering Institute, Tianjin University, China
ZAN WANG and HANMO YOU, College of Intelligence and Computing, Tianjin University, China
LINGMING ZHANG, Department of Computer Science, The University of Texas at Dallas, USA
MING YAN, College of Intelligence and Computing, Tianjin University, China

Deep neural network (DNN) has become increasingly popular and DNN testing is very critical to guarantee the correctness of DNN, i.e., the accuracy of DNN in this work. However, DNN testing suffers from a serious efficiency problem, i.e., it is costly to label each test input to know the DNN accuracy for the testing set, since labeling each test input involves multiple persons (even with domain-specific knowledge) in a manual way and the testing set is large-scale. To relieve this problem, we propose a novel and practical approach, called PACE (which is short for **P**ractical **AC**curacy **E**stimation), which selects a small set of test inputs that can precisely estimate the accuracy of the whole testing set. In this way, the labeling costs can be largely reduced by just labeling this small set of selected test inputs. Besides achieving a precise accuracy estimation, to make PACE more practical it is also required that it is interpretable, deterministic, and as efficient as possible. Therefore, PACE first incorporates clustering to interpretably divide test inputs with different testing capabilities (i.e., testing different functionalities of a DNN model) into different groups. Then, PACE utilizes the MMD-critic algorithm, a state-of-the-art example-based explanation algorithm, to select prototypes (i.e., the most representative test inputs) from each group, according to the group sizes, which can reduce the impact of noise due to clustering. Meanwhile, PACE also borrows the idea of adaptive random testing to select test inputs from the minority space (i.e., the test inputs that are not clustered into any group) to achieve great diversity under the required number of test inputs. The two parallel selection processes (i.e., selection from both groups and the minority space) compose the final small set of selected test inputs. We conducted an extensive study to evaluate the performance of PACE based on a comprehensive benchmark (i.e., 24 pairs of DNN models and testing sets) by considering different types of models (i.e., classification and regression models, high-accuracy and low-accuracy models, and CNN and RNN models) and different types of test inputs (i.e., original, mutated, and automatically generated test inputs). The results demonstrate that PACE is able to precisely estimate the accuracy of the whole testing set with only 1.181%~2.302% deviations, on average, significantly outperforming the state-of-the-art approaches.

This work was partially supported by the National Natural Science Foundation of China under Grant No. 61872263, Intelligent Manufacturing Special Fund of Tianjin under Grant No. 20191012, Innovation Research Project of Tianjin University under Grant No. 2020XZC-0042. This work was also partially supported by National Science Foundation under Grant No. CCF-1763906 and Alibaba.

Authors' addresses: J. Chen, Z. Wang (corresponding author), H. You, and M. Yan, College of Intelligence and Computing, Tianjin University, Tianjin, 300350, China; emails: {junjiechen, wangzan, youhanmo, yanming}@tju.edu.cn; Z. Wu, Tianjin International Engineering Institute, Tianjin University, Tianjin, 300350, China; email: wuzhuo@tju.edu.cn; L. Zhang, Department of Computer Science, The University of Texas at Dallas, Richardson, TX, 75080, USA; email: lingming.zhang@utdallas.edu.

1049-331X/2020/10-ART30 $15.00
https://doi.org/10.1145/3394112

CCS Concepts: • **Software and its engineering → Software testing and debugging**; • **Computer systems organization** → *Neural networks*;

Additional Key Words and Phrases: Deep neural network testing, test input selection, labeling, test optimization

ACM Reference format:
Junjie Chen, Zhuo Wu, Zan Wang, Hanmo You, Lingming Zhang, and Ming Yan. 2020. Practical Accuracy Estimation for Efficient Deep Neural Network Testing. *ACM Trans. Softw. Eng. Methodol.* 29, 4, Article 30 (October 2020), 35 pages.
https://doi.org/10.1145/3394112

1 INTRODUCTION

In recent years, deep neural network (DNN) has been widely studied in many domains and has gained great success in practice, such as autonomous driving cars [11], face recognition [93], medical diagnosis [69], aircraft collision avoidance systems [43], and software engineering [17, 18, 52, 114]. Unfortunately, DNN still suffers from bugs like traditional software systems [41, 60–62, 73, 103], which can result in serious consequences, even disasters in safety-critical domains. For example, a pedestrian was killed by an Uber autonomous driving car in Tempe, Arizona in 2018.[1] Also, a Tesla Model S in autopilot mode crashed into a parked fire truck with lights flashing on a California freeway in 2018.[2] Therefore, guaranteeing the quality of DNN is very critical.

In practice, DNN testing is one of the most effective ways to guarantee the quality of DNN. However, it suffers from two main challenges. First, it is difficult to ensure whether the used test inputs are sufficient enough to test DNN, since the practical scenario is very complex and changeable. To solve this challenge, researchers have paid much attention by proposing various metrics to measure the adequacy of test inputs [54, 60, 73] or designing various approaches to generating adversarial inputs [63, 70, 103]. Second, it is costly to label these test inputs to know the DNN accuracy for the testing set due to the following reasons: (1) Manual labeling is the main method, and usually labeling one test input involves multiple persons to ensure the labeling correctness; (2) the testing set is large-scale; (3) in many cases, labeling also relies on domain-specific knowledge, and thus it makes labeling more expensive by employing the persons with the domain-specific knowledge. According to our industrial partners, the second challenge is even more troublesome than the first one, but currently few efforts have been devoted to relieving this challenge, which is the target of our work.

Recently, Li et al. [55] proposed the first approach, called Cross Entropy-based Sampling (CES), which selects a small set of test inputs that can precisely estimate the accuracy of the whole testing set. In this way, the labeling cost can be reduced by just labeling this small set of test inputs. More specifically, CES conducts selection by minimizing the cross entropy between the selected set and the whole testing set. Although CES has been evaluated to be effective in the existing study [55], it suffers from several limitations in practice. First, the selection result produced by CES does not have good interpretability. It is hard to explain the relation between the selected test inputs and the testing goal (e.g., testing various functionalities of a DNN model). Second, CES involves randomness, and thus it can produce different selection results at different runs for the same DNN model and the achieved effectiveness by different selection results has fluctuation to some degree. Third, the effectiveness of CES still needs to be improved. Therefore, a practical approach to selecting a small number of test inputs to precisely estimate the accuracy of the whole testing set is still urgently desirable.

[1] https://www.vice.com/en_us/article/9kga85/uber-is-giving-up-on-self-driving-cars-in-california-after-deadly-crash.
[2] https://www.newsweek.com/autonomous-tesla-crashes-parked-fire-truck-california-freeway-789177.

To improve the efficiency of DNN testing, in this article, we propose a novel and practical approach for test input selection to reducing labeling cost, called **PACE** (**P**ractical **AC**curacy **E**stimation). To make PACE practical, it is required that PACE is *interpretable*, *deterministic*, and *efficient* as much as possible. Moreover, it is required that PACE is able to select a small set of test inputs to *precisely* estimate the accuracy of the whole testing set. To satisfy the above two requirements, PACE first clusters test inputs into different groups by identifying effective features, to discriminate test inputs with different testing capabilities (i.e., testing different functionalities of a DNN model). This is also a step to augment the interpretability. Then, PACE selects test inputs from each group to make this small set to be able to test various functionalities of a DNN model. Since it is hard to perfectly discriminate test inputs with different testing capabilities via clustering, we should select the test inputs that best represent the testing capability of the group to bypass the impact of noise in each group. Here, PACE adopts the MMD-critic algorithm [44], a state-of-the-art example-based explanation algorithm (which is used for the interpretability of a machine learning model including classification and clustering), to select prototypes (i.e., the most representative test inputs) from each group. This is another step to further augment the interpretability. Also, there may be some test inputs that do not belong to any group, indicating that each of them is likely to have unique testing capability. We call the space of these test inputs *minority space*. To better approximate the whole set, PACE not only selects test inputs from each group according to the group sizes to maintain the similar distribution of testing capabilities but also selects test inputs from the minority space. More specifically, PACE borrows the idea of adaptive random testing [22] for selection to explore the minority space with the limited number of test inputs as sufficiently as possible.

We conducted an extensive study to evaluate the effectiveness of PACE based on 24 pairs of DNN models under test and testing sets. In our benchmark, we considered both classification models and regression models, high-accuracy models and low-accuracy models, Convolutional Neural Network (CNN) models and Recurrent Neural Network (RNN) models, and different types of test inputs (i.e., original test inputs, mutated test inputs, and automatically generated test inputs). Our experimental results demonstrate that PACE is able to precisely estimate the accuracy of the whole testing set with only 1.181%~2.302% deviations on average for all the subjects, significantly outperforming all the three compared approaches (i.e., Simple Random Sampling (SRS), Cross Entropy-based Sampling (CES), and Confidence-based Stratified Sampling (CSS) to be introduced in Section 4.2) with the average improvements of 51.06%, 50.94%, and 70.12%, respectively. We further investigated the contribution of each component in PACE, and the results demonstrate that each component (including clustering, MMD-critic-based selection, and adaptive random selection) indeed contributes to PACE.

To sum up, our work has the following major contributions:

- **Approach.** We propose the first relatively interpretable, deterministic, and efficient approach, PACE, to selecting a subset of DNN test inputs for estimating the accuracy of the whole testing set.
- **Implementation.** We implement the proposed approach based on state-of-the-art frameworks and libraries, such as Keras 2.2.4 [3] with TensorFlow 1.14.0 [6], as well as HDBSCAN, FastICA, and MMD-critic algorithms provided by hdbscan 0.8.22 [2], sklearn [1], and the authors of the MMD-critic algorithm [4], respectively.
- **Study.** We conduct an extensive study based on 24 pairs of DNN models under test and testing sets, in which the diversity of both models and test inputs are considered carefully, demonstrating the effectiveness of PACE.

- **Artifact.** We have released an extensive dataset for future usage and research, including our implementation as well as experimental data.

2 BACKGROUND

2.1 DNN and DNN Testing

DNN is composed of multiple layers and each layer contains a large number of neurons [57]. The neurons between layers are connected with links equipped with weights. The weights are acquired based on the training process with training data. DNN maps inputs to outputs via calculation based on these weights. Currently, DNN is mainly divided into Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN). CNN involves convolution computing and is usually used to deal with data with grid-like topology (e.g., images) [29], while RNN uses loops to keep learned knowledge and is often used to deal with sequential data (e.g., natural language and speech) [47].

DNN testing is one of the most widely used ways of guaranteeing the quality of DNN [60, 73]. Like traditional software systems, DNN testing also involves test inputs and test oracles. Test inputs in DNN testing refer to the inputs that need to be predicted by DNN. According to the specific task of the DNN under test, the test input can be image, natural language, or speech. Test oracles in DNN testing are based on manual labeling. That is, it is required for each test input to manually label its ground-truth by persons. By comparing the labeled ground truth and the predicted result, it is clear to determine whether a test input is predicted correctly by the DNN model.

2.2 Test Optimization in DNN Testing

As presented in Section 1, it is costly to label test inputs. To improve the efficiency of DNN testing, there are two categories of methods to optimize the testing process. The first one is test input selection. It aims to select a small set of test inputs that can precisely estimate the accuracy of the whole testing set. Then, developers can just label the small set of test inputs instead, to save the labeling costs. Li et al. [55] proposed the state-of-the-art approach in this category, which selects test inputs by minimizing the cross entropy between the selected set and the whole testing set based on the outputs of the last hidden layer of the DNN under test. The second category is test input prioritization. It aims to rank all the test inputs based on their probabilities that can be incorrectly predicted by the DNN under test *without discarding any test input*. In this way, developers can find the test inputs revealing incorrect behaviors earlier. Shi et al. [91] proposed an approach for test input prioritization by measuring the purity of test inputs following the idea of Gini purity [75] based on the confidences outputted by the DNN. Zhang et al. [111] proposed to rank test inputs by calculating their noise sensitivity. By adding the same noise to the test inputs, the test inputs with higher noise sensitivity are more likely to fool the DNN than those with lower noise sensitivity. Furthermore, Ma et al. [64] proposed a set of metrics based on model confidence on specific inputs to select test inputs that are more likely to be misclassified, which has the similar goal with test input prioritization above.

Our work belongs to the first category, i.e., test input selection, to estimate the accuracy of the whole testing set using a small set of selected test inputs, and we discuss and compare it with the state-of-the-art test input selection approaches [55] in Sections 4 and 5.

3 APPROACH

To improve the efficiency of DNN testing, we aim to select a subset of test inputs to represent the whole testing set. That is, we hope to precisely estimate the accuracy of the whole set by only labeling the small set of selected test inputs. In this way, the costs of labeling can be reduced largely. However, how to effectively select such a small set of test inputs is challenging. Moreover,

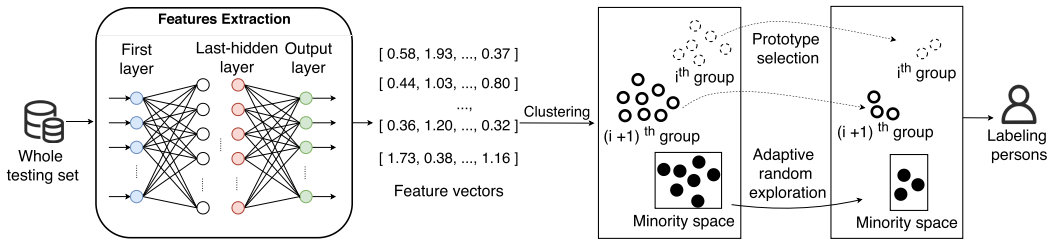


Fig. 1. Overview of PACE.

it is required that the selection approach is interpretable, efficient, and deterministic as much as possible, to make it applicable in practice. This is also a critical challenge to solve this problem.

In this article, we propose a novel and practical approach, called **PACE** (**P**ractical **AC**curacy **E**stimation), to selecting such a small set of test inputs. In general, for a whole testing set, some test inputs have similar testing capabilities (i.e., testing the similar functionalities of a DNN) while some test inputs have different testing capabilities. Intuitively, the small set of selected test inputs should cover various testing capabilities and maintain the original distribution of these testing capabilities, to represent the whole set as much as possible with high interpretability. To achieve this goal, PACE clusters all the test inputs into groups based on the identified features reflecting their testing capabilities from them, in which different groups are more likely to have different testing capabilities. Since the sizes of groups reflect the distribution of different testing capabilities to some degree, PACE selects the required number of test inputs according to the proportion of different group sizes, to maintain the similar distribution of testing capabilities.

A natural follow-up question is which test inputs should be selected from each group. Since it is difficult to perfectly discriminate test inputs with different testing capabilities via clustering, we should select the test inputs that best reflect the testing capability of the group to bypass the impact of noise in each group. To further augment the interpretability of the selected test inputs, PACE utilizes the MMD-critic algorithm [44], a state-of-the-art example-based explanation algorithm, to select prototypes (i.e., the most representative test inputs) from each group. Furthermore, there may be also some test inputs that do not belong to any group. That is, each of them is likely to have unique testing capability. We call the space of these test inputs *minority space*. It is reasonable to select test inputs from the minority space as well, to further approximate the whole set. To more sufficiently explore the minority space using the required number of test inputs, PACE borrows the idea of adaptive random testing [22] to select test inputs.

Figure 1 presents the overview of our approach PACE. In the following, we first introduce the studied features of test inputs for clustering in Section 3.1, and then present clustering-based testing capability discrimination in Section 3.2. Next, we present MMD-critic-based prototype selection in Section 3.3 and adaptive random exploration for minority space in Section 3.4. Finally, we summarize the usage of PACE in Section 3.5.

3.1 Studied Features

We first identify features of test inputs to help discriminate their testing capabilities. Following traditional software testing [20, 21, 110], there are two types of features that can reflect testing capabilities to some degree, i.e., coverage features and input features. For DNN testing, the former refers to which DNN elements are covered when executing a test input while the latter does not rely on coverage information but relies on test-input information. In this work, we use *input features* to help discriminate testing capabilities. There are three main reasons: (1) The existing work [91] has demonstrated that, many test inputs may have very similar neuron coverage for a given DNN and

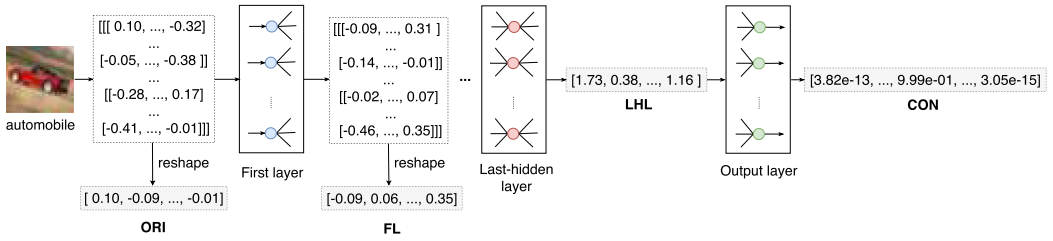


Fig. 2. Feature extraction visualization.

thus coverage features cannot discriminate their testing capabilities well. (2) The effectiveness of coverage features has been evaluated in the problem of selecting test inputs and the results demonstrated that its effectiveness is worse than that of input features [55]. (3) Collecting coverage features tend to incur extra costs.

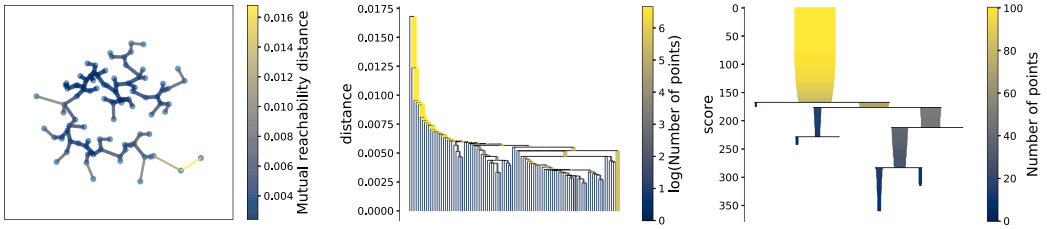
A DNN gradually learns features from inputs to predict labels. Different layers of a DNN represent different types of input features. The layers more close to the input layer represent more basic features while the layers more close to the output layer represent more high-order features. That is, test-input information contains basic information, i.e., the test input itself and the basic features extracted from the test input, and high-level information, i.e., the high-order features extracted from the test input. More high-order features can more precisely capture the relations between inputs and labels, but they are more specific to a given DNN model and collecting them is more costly, since it needs to run more layers of a DNN model. In contrast, more basic features are more general and can be collected more efficiently, but they cannot reflect more complex patterns of input features to predict labels. Our approach is not specific to certain types of input features, and we study different types of input features in this work. More specifically, we study the following four types of input features (including both basic features and high-order features) as the representatives:

- **Original features (ORI):** refer to the input vector of a DNN, which are the most basic features and directly represent an input itself.
- **First-layer features (FL):** refer to the output of the first layer in a DNN, which are the most close one to the input vector of a DNN.
- **Last-hidden-layer features (LHL):** refer to the output of the last hidden layer in a DNN, which are the high-order features that can directly infer the prediction result for an input.
- **Confidence features (CON):** refer to the output of a DNN (classifier), which represent the confidence of a prediction result and have been used by the existing work [55, 91]. We also classify it to input features, since it does not rely on coverage information but relies on the output derived from a test input via a DNN model.

Figure 2 illustrates where each type of features come from in DNN using one input example (i.e., an *automobile* image in *CIFAR-10* used in our study to be presented in Section 4). Please note that, the input of a DNN and the output of some layer in a DNN may be a multidimensional matrix, and thus we reshape the matrix to a vector as a feature vector (such as ORI and FL in Figure 2). In particular, ORI features are static features, since they do not need to run a DNN, while FL, LHL, and CON features are all dynamic features.

3.2 Clustering-based Testing Capability Discrimination

Based on the identified features of test inputs (e.g., ORI or FL), each test input is represented as a feature vector. PACE clusters test inputs into different groups to discriminate their testing



(a) Minimum spanning tree (b) Connected component hierarchy (c) Condensed tree

Fig. 3. Visualization of main stages in HDBSCAN clustering.

capabilities based on the set of feature vectors. Before clustering, we first normalize features to adjust the values measured on different scales to a common scale. Since the features are all numeric type, PACE adopts min-max normalization [19, 25] to adjust each value of these features into the interval [0,1]. Suppose a whole testing set is denoted as T whose size is denoted as st, the set of test inputs in T is denoted as $T = \{t_1, t_2, \ldots, t_{st}\}$ and the feature vector of t_i is denoted as $F_i = \{f_{i1}, f_{i2}, \ldots, f_{ir}\}$, we use a variable x_{ij} to represent the value of the jth feature for t_i before normalization and use a variable x^*_{ij} to represent the value of the jth feature for t_i after normalization ($1 \leq i \leq st$ and $1 \leq j \leq r$). Equation (1) shows the calculation of normalization:

$$x^*_{ij} = \frac{x_{ij} - min(\{x_{kj} | 1 \leq k \leq st\})}{max(\{x_{kj} | 1 \leq k \leq st\}) - min(\{x_{kj} | 1 \leq k \leq st\})}. \quad (1)$$

PACE then adopts the HDBSCAN (Hierarchical Density-based Spatial Clustering of Applications with Noise) algorithm [66] to cluster test inputs due to the following reasons. (1) It is scarcely possible to know the number of types of testing capabilities in advance, and thus the clustering algorithms that are required to pre-define the number of clusters cannot be applicable, such as the widely used K-means algorithm [37]. HDBSCAN does not need to pre-define the number of clusters and performs clustering based on density. (2) One of the most widely used density-based clustering algorithms is DBSCAN [85], and HDBSCAN is its upgraded version. In particular, HDBSCAN can have varying density clusters while DBSCAN has to pre-define the density of clusters. (3) HDBSCAN has been demonstrated to be very efficient [66], making our approach PACE more practical. (4) HDBSCAN has few parameters to set and is robust to parameter selection [66], which makes it much easier to be used in practice.

HDBSCAN is a density-based clustering algorithm that groups the data points that are closely packed together (i.e., data points with many nearby neighbors). To more clearly illustrate the process of HDBSCAN in PACE, we sample 100 test inputs from the testing set *CIFAR-10* of the DNN model *ResNet-20* (used in our study to be presented in Section 4) to make visualization for main stages in HDBSCAN clustering, which is shown in Figure 3. More specifically, it first constructs a weighted graph, where data points are vertices and there is an edge between any two points whose weight is equal to the mutual reachability distance between the two points. Equation (2) shows the calculation of the mutual reachability distance (MRD) between points a and b based on K nearest neighbor:

$$MRD_k(a, b) = max\{Core_k(a), Core_k(b), Dist(a, b)\}, \quad (2)$$

where Core$_k$(a) and Core$_k$(b) represent the distance between a/b and its kth nearest neighbor, respectively; Dist(a, b) is the distance between a and b, measured by the widely used *Euclidean distance*. Under MRD, dense points remain the same distance from each other, but sparser points are pushed away to be at least their core distance away from any other point. Then, HDBSCAN

builds the minimum spanning tree of the weighted graph via Prim's algorithm [68], as shown in Figure 3(a). Based on the minimum spanning tree, a completely connected graph is transformed to a hierarchy of connected components by ranking the tree edges in the ascending order of the distances and iterating via creating a new merged cluster for each edge, as shown in Figure 3(b). Next, it condenses down the large cluster hierarchy into a smaller tree based on minimum cluster size, which is one parameter of HDBSCAN, as shown in Figure 3(c). Finally, it extracts the stable clusters from the condensed tree by calculating the stability score of each cluster. Based on HDBSCAN, PACE divides test inputs into different groups and different groups are more likely to have different testing capabilities. Moreover, some test inputs are not clustered into any group and constitute the *minority space*.

Please note that, when the dimension of features increases, the performance of HDBSCAN could be largely decreased [66]. Therefore, PACE incorporates the process of dimension reduction for high-dimensional features before clustering. More specifically, PACE uses the FastICA algorithm [71] to perform dimension reduction. FastICA aims to find independent components and is helpful to find underlying factors, by maximizing the negative entropy defined by Equation (3). In this formula, Y_{Gauss} is a Gaussian random variable with the same variance as a random variable Y, $E[.]$ is to calculate the mean values, and $g(.)$ is a nonlinear function used to approximate the differential entropy:

$$N_g(Y) = \{E[g(Y) - E[g(Y_{Gauss})]]\}^2. \tag{3}$$

3.3 MMD-critic-based Prototype Selection

After dividing test inputs with different testing capabilities into different groups, PACE then selects test inputs from each group to constitute the small set of test inputs. Since it is hard to perfectly discriminate different testing capabilities via clustering, each group could have noise, i.e., the test inputs that should not be divided into this group. To bypass the impact of noise in each group, it is interpretable to select the test inputs that can best represent the testing capability of the group, which are also called *prototypes*. To augment the interpretability of the selected test inputs from each group, PACE utilizes the MMD-critic algorithm [44], a state-of-the-art example-based explanation algorithm (which is used for the interpretability of a machine learning model including classification and clustering), to select prototypes from a group.

More specifically, it selects prototypes by calculating the difference between the prototype distribution (denoted as P) and the group distribution (denoted as G). The selected prototypes should have the minimum difference. The MMD is a measure of the difference between P and G, given by the supremum over a function space $\mathcal{F}$ of the differences between the expectations with respect to the two distributions, which is shown in Equation (4). Besides, PACE also selects criticisms, which are the test inputs differing the most from the prototypes in the group, based on the MMD-critic algorithm. The criticisms together with the prototypes are able to help developers build a better mental model to understand the group, and thus the interpretability of the selection can be improved:

$$\text{MMD}(\mathcal{F}, P, G) = \sup_{f \in \mathcal{F}} (\mathbb{E}_{X \sim P}[f(X)] - \mathbb{E}_{Y \sim G}[f(Y)]). \tag{4}$$

In this way, PACE selects the required number of prototypes from each group to constitute the small set, and also selects the corresponding criticisms to help understand the complex test-input space in the group.

3.4 Adaptive Random Exploration for Minority Space

The test inputs in the minority space are also the constituent part of the whole testing set, and thus the selected small set should also contain a part of these test inputs, to better represent the

whole testing set. However, it is challenging to select a part of these test inputs to represent the whole minority space, since these test inputs are likely to have different testing capabilities with each other. To explore the minority space as sufficiently as possible using the required number of test inputs, PACE borrows the idea of adaptive random testing [22]. In this way, PACE can achieve great diversity of testing capabilities under the required number of test inputs so that the whole minority space can be represented by the required number of test inputs as much as possible. More specifically, it calculates the distance between an unselected test input and each already selected test input, and uses the minimum distance as the distance of the unselected test input with the already selected test inputs. Then, it selects the test input that has the maximum distance with the already selected test inputs as the next one. This adaptive-random strategy has been demonstrated to be more effective than other adaptive-random strategies in the existing study [42]. Following the used distance in clustering (presented in Section 3.2), PACE also uses *Euclidean Distance* to calculate the distance between test inputs. Equation (5) presents the calculation at each time of selection, where U and S represent the set of unselected test inputs and the set of already selected test inputs, respectively, and *EucDist(,)* aims to calculate the Euclidean distance between two test inputs:

$$d = \max_{y \in U}\{\min_{x \in S} EucDist(x, y)\}. \tag{5}$$

Traditional adaptive random testing randomly selects the first sample, but PACE selects the first test input determinately to make it more practical and interpretable. More specifically, PACE selects the test input that has the most unique testing capability as the first one, which has the largest distance with all the groups. In particular, this test input is the one at the Top-1 position by ranking the test inputs in the minority space based on their *outlier_scores_* (measuring the distance with all the groups) values provided by the HDBSCAN clustering algorithm.

3.5 Usage of PACE

In this subsection, we present the usage of PACE and Algorithm 1 shows high-level pseudo code of PACE. The input of PACE includes a DNN under test denoted as D, a whole testing set denoted as T whose size is denoted as st (as presented in Section 3.2), and a required number representing the size of the small set of selected test inputs denoted as n. To construct a small set of test inputs that can precisely estimate the accuracy of the whole set, PACE first transforms each test input into a feature vector by extracting features (e.g., ORI or FL), shown in Lines 2–4 in Algorithm 1. Then, PACE pre-processes the set of vectors including normalization or dimension reduction, and divides them into m groups and the minority space by clustering (Lines 5 and 6 in Algorithm 1). Here, we denote the size of the ith group as s_i and denote the size of the minority space as sm, where $\sum_{i=1}^{m} s_i + sm = st$. The selected small set of test inputs should contain both test inputs from each group and those from the minority space. Here, we define a threshold α to determine the proportional distribution of them. That is, the number of selected test inputs from groups is $\alpha \cdot n$ and the number of selected test inputs from the minority space is $(1 - \alpha) \cdot n$ (which is smaller than sm). Further, PACE selects test inputs from each group according to the proportion of different group sizes to maintain the similar distribution of testing capabilities. That is, the number of selected test inputs from the kth group is $\frac{s_k}{\sum_{i=1}^{m} s_i} \cdot (\alpha \cdot n)$. After determining the number of selected test inputs from each group or the minority space, PACE selects them based on the MMD-critic-based prototype selection method (Lines 7–11 in Algorithm 1) or the adaptive random selection method (Lines 12–14 in Algorithm 1), and finally constructs the small set of test inputs, which is the output of PACE (Line 15 in Algorithm 1). Developers can just label this small set of test inputs to estimate the accuracy of the whole testing set in practice.

ALGORITHM 1: High-level pseudo code of PACE

Input: D: the DNN model under test
T: the whole testing set, whose size is st
n: the required number of selected test inputs from T
Output: $\mathcal{X}$: a set of selected test inputs, whose size is n

$\mathcal{X} = \Phi$
foreach t_i *in* T **do**
  $f_i \leftarrow extractFeatures(t_i, D)$/* transforming a test input to a feature vector */
end
$\{f'_1, f'_2, \ldots, f'_{st}\} \leftarrow Process(f_1, f_2, \ldots, f_{st})$/* pre-processing these feature vectors */
$\{g_1, g_2, \ldots, g_m, \mathcal{M}\} \leftarrow cluster(f'_1, f'_2, \ldots, f'_{st})$/* clustering all the feature vectors into m groups $G \leftarrow \{g_1, g_2, \ldots, g_m\}$ and the minority space $\mathcal{M}$ */
foreach g_k *in* G **do**
  $selectedNumber \leftarrow \frac{s_k}{\sum_{j=1}^{m} s_j} \cdot (\alpha \cdot n)$/* determining the required number of selected test inputs from g_k */
  $selectedInput \leftarrow MMDselection(g_k, selectedNumber)$/* conducting MMD-critic-based selection from g_k */
  $\mathcal{X}.add(selectedInput)$
end
$selectedNumber \leftarrow (1 - \alpha) \cdot n$/* determining the required number of selected test inputs from $\mathcal{M}$ */
$selectedInput \leftarrow ARTselection(\mathcal{M}, selectedNumber)$/* conducting adaptive random selection from $\mathcal{M}$ */
$\mathcal{X}.add(selectedInput)$
return $\mathcal{X}$

4 EVALUATION DESIGN

In this study, we aim to address the following five research questions:

- **RQ1:** How does PACE perform in terms of effectiveness and efficiency?
- **RQ2:** What is the impact of different features on PACE?
- **RQ3:** What is the impact of the threshold α on PACE?
- **RQ4:** What is the impact of dimension reduction (including different dimension numbers and different dimension reduction algorithms) on PACE?
- **RQ5:** Does each component contribute to PACE?

4.1 DNN Models and Datasets

In our study, we used 24 pairs of DNN models under test and testing sets as subjects in total. The used DNN models are trained based on eight popular datasets (i.e., MNIST, CIFAR-10, CIFAR-100, SVHN, Driving, Fashion-MNIST, ImageNet, and Speech-Commands), respectively, which have been widely used in the existing studies [55, 91, 99]. More specifically, MNIST is a handwritten digit dataset,[3] CIFAR-10 is a 10-class ubiquitous object dataset,[4] CIFAR-100 is a 100-class ubiquitous object dataset,[5] SVHN is a street view house number dataset collected from real-world scenes,[6] Driving is an autonompus driving dataset provided by Udacity,[7] Fashion-MNIST is a 10-class product dataset provided by the research department of Zalando, a German fashion technology company,[8] ImageNet is an image dataset organized according to the

[3] http://yann.lecun.com/exdb/mnist/.
[4] http://www.cs.toronto.edu/~kriz/cifar.html.
[5] http://www.cs.toronto.edu/~kriz/cifar.html.
[6] http://ufldl.stanford.edu/housenumbers/.
[7] https://udacity.com/self-driving-car.
[8] https://github.com/zalandoresearch/fashion-mnist.

Table 1. DNN Models and Testing Sets

ID	Testing set	Model	Size(KB)	#Tests	Accuracy(%)	Task	Test Type
1	MNIST	LeNet-1	113	10,000	94.86	classification	original
2		LeNet-4	947	10,000	96.79	classification	
3		LeNet-5	1,093	10,000	98.68	classification	
4		LeNet-5-M1	1,093	10,000	79.53	classification	
5		LeNet-5-M2	1,093	10,000	77.27	classification	
6		LeNet-5-M3	1,093	10,000	79.14	classification	
7	CIFAR-10	VGG-16	21,814	10,000	96.07	classification	original
8		ResNet-20	3,507	10,000	91.45	classification	
9	CIFAR-100	ResNet-20	10,615	10,000	71.42	classification	original
10	SVHN	LeNet-5	522	26,032	87.90	classification	original
11	Driving	Dave-orig	8,306	5,614	90.34	regression	original
12		Dave-drop	12,832	5,614	91.82	regression	
13	Driving-patch	Dave-orig	8,306	5,614	74.18	regression	mutated
14		Dave-drop	12,832	5,614	71.88	regression	
15	Driving-light	Dave-orig	8,306	5,614	84.98	regression	mutated
16		Dave-drop	12,832	5,614	71.17	regression	
17	Fashion-MNIST	LeNet-5	385	10,000	89.88	classification	original
18	Autogen-MNIST	LeNet-5	1,093	10,000	49.35	classification	generated
19	Autogen-CIFAR-10	ResNet-20	3,507	10,000	42.92	classification	generated
20	Autogen-SVHN	LeNet-5	522	10,000	43.69	classification	generated
21	Autogen-Fashion	LeNet-5	385	10,000	45.31	classification	generated
22	ImageNet	VGG-19	562,176	50,000	64.73	classification	original
23	ImageNet	ResNet-50	100,352	50,000	68.27	classification	original
24	Speech-Commands	DeepSpeech	6,734	6,417	94.53	classification	original

*The accuracy of ImageNet models in our study is slightly smaller than that in Reference [92], since the pre-processing methods used in our study and that paper are slightly different. More specifically, to obtain the fixed-size 224×224 input images from ImageNet, that paper randomly crops images from rescaled images, while we resize images without cropping following the method provided by official Keras examples.

WordNet hierarchy, which is more realistic and complex,[9] and Speech-Commands is a sequential dataset, which contains a set of one-second .wav audio files collected using crowdsourcing, each containing a single spoken English word.[10] Table 1 presents the detailed information of the used models and their testing sets. In this table, the last five columns represent the size of the model, the size of the whole testing set for the model, the accuracy achieved by the whole testing set for the model, the task of the model (classification or regression), and the type of test inputs (original, mutated, or automatically generated test inputs), respectively. Since the practical scenario is very complex, we tried to construct a comprehensive benchmark in our study.

First, we considered different types of test inputs: the original test inputs, the mutated test inputs (i.e., Driving-patch and Driving-light), and the automatically generated test inputs (i.e., Autogen-MNIST, Autogen-CIFAR-10, Autogen-SVHN, and Autogen-Fashion). The mutated test inputs aim to simulate the different physical environments when a model is applied in practice. Following the existing work [55], for the models Dave-orig and Dave-drop trained based on Driving, we produced Driving-patch by randomly blocking some parts of each test input in Driving to simulate

[9] http://www.image-net.org.
[10] https://github.com/bjtommychen/Keras_DeepSpeech2_SpeechRecognition.

block some parts of a camera, and produced Driving-light by randomly changing the intensities of lights for each test input in Driving. Currently, automatically generating testing inputs (also called adversarial examples) is also a popular method to test a DNN model. Therefore, we also used the automatically generated test inputs in our study. Following the existing work [91], we used the Basic Iterative Method [46] to automatically generate test inputs. More specifically, based on MNIST, CIFAR-10, SVHN, and Fashion-MNIST, we constructed Autogen-MNIST, Autogen-CIFAR-10, Autogen-SVHN, and Autogen-Fashion, respectively, each of which contains 5,000 test inputs generated automatically and 5,000 test inputs selected randomly from the original testing set, following the practice of the existing work [91].

Second, we considered the DNN models with different accuracy, i.e., high-accuracy models and low-accuracy models. Here, we define the model whose accuracy is smaller than 80% as a low-accuracy model; otherwise, the model is a high-accuracy model. The existing work [55] produced mutated models to simulate low-accuracy models. Following this existing work, we also adopted their mutated models (i.e., LeNet-5-M1, LeNet-5-M2, and LeNet-5-M3) in our study. More specifically, the three mutated models were produced based on three mutation strategies. The first mutation strategy is to exchange the labels "8" and "0" in the training data, and then build the mutated model LeNet-5-M1. The second one is to exchange the labels "7" and "1" in the training data, and then build the mutated model LeNet-5-M2. The third one is to exchange the labels "9" and "3" in the training data, and then build the mutated model LeNet-5-M3. Besides, we used three real-world low-accuracy models, i.e., ResNet-20 based on CIFAR-100, VGG-19 based on ImageNet, and ResNet-50 based on ImageNet, as models under test in our study. Also, the mutated test inputs and the automatically generated test inputs provided some low-accuracy models, including Dave-orig with Driving-patch, Dave-drop with Driving-patch, Dave-drop with Driving-light, ResNet-20 with Autogen-CIFAR-10, LeNet-5 with Autogen-SVHN, and LeNet-5 with Autogen-Fashion.

Third, we considered the DNN models with different tasks, i.e., classification models and regression models, in our study, where we adopts the regression models used in the existing work [55], i.e., Dave-orig and Dave-drop (ID: 11-16), and the other models used in our study are classification models. In particular, DeepSpeech is a multi-label classification model while the other classification models are single-label classification models. Moreover, we also considered both CNN and RNN models, where DeepSpeech is a RNN model while the other models are CNN models.

4.2 Independent Variables

In the study, we considered the following seven independent variables:

• **Compared Approaches.** In our study, we considered three compared approaches in total, including one baseline (i.e., SRS) and two state-of-the-art approaches (i.e., CES and CSS).

SRS (Simple Random Sampling) randomly selects a required number of test inputs from the whole testing set, where each test input has the same probability to be selected. SRS is regarded as the baseline in our study.

CES (Cross Entropy-based Sampling) [55] selects a required number of test inputs by minimizing the cross entropy between the selected set and the whole testing set to guarantee the distribution similarity. In particular, CES transforms each test input into a feature vector by extracting its last-hidden-layer features. CES is the state-of-the-art approach to selecting a small set of test inputs to estimate the accuracy of the whole testing set.

CSS (Confidence-based Stratified Sampling) [55] is based on the confidence features of test inputs. CSS first divides the confidence values into different intervals, and then selects test inputs based on their confidence values in different intervals, to guarantee the distribution similarity between the selected set and the whole testing set. CSS is also regarded as a state-of-the-art approach.

The existing work [55] has compared the three approaches, and the experimental results demonstrated that in general CES is the most effective approach among them. However, CSS performs better than CES for high-accuracy models, while the former performs worse than the latter, even worse than SRS for low-accuracy models, since the confidence is not reliable for low-accuracy models. Also, CSS cannot be applied to regression models and multi-label classification models, while CES and SRS can be applied to regression models, single-label classification models, and multi-label classification models. That is, in our study CSS cannot be applied to Dave-orig, Dave-drop, and DeepSpeech. In particular, all the three compared approaches involve randomness, and thus we repeated them 50 times following the existing work [55].

• **Variants of PACE.** To investigate the contribution of each component in PACE, we proposed four variants of PACE as follows.

PACE$^{g}_{rand}$ replaces the MMD-critic-based selection from the groups with random selection in PACE. That is, PACE$^{g}_{rand}$ randomly selects test inputs from each group after clustering. This variant aims to investigate the contribution of the MMD-critic-based selection in PACE.

PACE$^{m}_{rand}$ replaces the adaptive random selection from the minority space with random selection in PACE. That is, PACE$^{m}_{rand}$ randomly selects test inputs from the minority space. This variant aims to investigate the contribution of the adaptive random selection in PACE.

To investigate the contribution of clustering in PACE, we proposed **PACE$_{mmd}$** and **PACE$_{art}$**. The former directly selects test inputs based on the MMD-critic algorithm from the whole testing set without clustering, while the latter directly selects test inputs based on the adaptive random method from the whole testing set without clustering.

PACE$^{g}_{rand}$, PACE$^{m}_{rand}$, and PACE$_{art}$ involve randomness, and thus they were repeated 50 times. Here, PACE$_{art}$ randomly selects the first test input.

• **Number of Selected Test Inputs.** Similar to the existing work [55], we set different sizes of the selected testing set, i.e., ranging from 50 to 180 with the interval of 10, to investigate the effectiveness of PACE more sufficiently.

• **Features.** We studied four types of features in our study, including ORI, FL, LHL, and CON (presented in Section 3.1). Here, we used LHL as the default feature in PACE.

• **Threshold α Values.** We investigated the impact of the threshold α in our study. Intuitively, the number of selected test inputs from the minority space should be smaller than that from the clustered groups, and thus we considered the α value to be 0.5, 0.6, 0.7, 0.8, and 0.9, respectively. Here, we used 0.8 as the default setting in PACE.

• **Dimension Numbers.** We investigated the impact of different dimension numbers of the FastICA algorithm on PACE. Here, we studied four different dimension numbers, i.e., 2, 4, 8, and 16, and our result recommends 2 as the default number of the FastICA algorithm in PACE.

• **Dimension Reduction Algorithms.** We investigated the impact of different dimension reduction algorithms on PACE. Here, we studied four popular dimension reduction algorithms, including the PCA algorithm, the NMF algorithm, the FA algorithm, and our currently used FastICA algorithm. More specifically, the PCA (Principal Component Analysis) algorithm projects the data to a lower dimensional space by keeping only the most significant singular vectors [101]. The NMF (Non-Negative Matrix Factorization) algorithm conducts dimension reduction by finding two non-negative matrices whose product approximates a non-negative matrix via factorization [48]. The FA (Factor Analysis) algorithm [83] aims to identify certain unobservable factors from the observed variables to conduct dimension reduction. In our study, we set the dimension number

of all these algorithms to be 2 and used the default settings of the other parameters provided by sklearn [5].

4.3 Measurements

The goal of PACE is to estimate the accuracy of the whole testing set using a small set of test inputs, and thus following the existing work [55], we used *Mean-squared Errors (MSE)* to measure the effectiveness of PACE, the compared approaches, and the variants of PACE. Although the goal of DNN testing also contains "revealing potential errors" and "covering the corner cases," these actually are the target of test input prioritization as presented in Section 2.2 rather than test input selection that our work focuses on. Therefore, we did not use these measurements to evaluate the effectiveness of these test input selection approaches in the study. Since some approaches were repeated 50 times due to randomness, the calculation of the MSE for them is shown in Equation (6). In this equation, $a\hat{c}c_i$ and acc refer to the estimated and actual accuracy, respectively. PACE and PACE_{mmd} are also calculated based on Equation (6), but they evaluate to $|a\hat{c}c - acc|$, since they are deterministic and we ran them only once. The smaller the MSE value is, the better the effectiveness of a selection approach is:

$$MSE = \sqrt{\frac{1}{50}\sum_{i=1}^{50}|a\hat{c}c_i - acc|^2}. \tag{6}$$

Besides, we also measured the efficiency of each selection approach by recording the time cost spent on selection.

4.4 Implementations

We implemented PACE using Python and extracted features based on Keras 2.2.4 [3] with TensorFlow 1.14.0 [6]. We adopted the implementations of the HDBSCAN algorithm, various dimension reduction algorithms, and the MMD-critic algorithm provided by hdbscan 0.8.22 [2], sklearn [1], and the authors of the MMD-critic algorithm [4], respectively. For the parameters in HDBSCAN, we set *min_cluster_size* to be 80 and *min_samples* to be 4 for *all the subjects* in our study, based on a small dataset. As demonstrated by the existing work [66], HDBSCAN is robust to parameter selection. The performance of HDBSCAN could be largely decreased when the dimension of features is high, and thus PACE performs dimension reduction before clustering in this case based on FastICA. More specifically, when the number of produced groups by HDBSCAN is very small (i.e., no more than 3) or a large portion of test inputs (i.e., more than 80% test inputs) are divided into the same group, indicating the poor performance of clustering, PACE conducts dimension reduction for them. Here, we also implemented a tool to automatically determine the application of dimension reduction. For the compared approach SRS, CES, and CSS, we adopted the existing implementations released by the existing work [55].

Our experiments are conducted on the Intel Xeon E5-2640 machine with 128GB RAM, CentOS 7.6. All the code and data in our work are available at the project homepage: https://github.com/pace2019/pace.

4.5 Process

We present the process of our study for each model under test and its whole testing set as follows.

First, we ran each selection approach (including PACE, three compared approaches, and four variants of PACE) to produce a set of selected test inputs, under the size ranging from 50 to 180 with the interval of 10. The small set of selected test inputs are used to estimate the accuracy of the whole testing set. For the approaches involving randomness, we ran them 50 times. We then calculated the MSE value and the time cost spent on selection for each approach. Based on these

results, we can answer RQ1 (comparing PACE with SRS, CES, and CSS) and RQ5 (comparing PACE with its four variants).

To investigate the effectiveness of different features in RQ2, we ran PACE by using ORI, FL, and CON to select test inputs under different sizes, respectively. We then calculated the MSE value for PACE with each type of feature.

To investigate the impact of the threshold α on PACE in RQ3, we ran PACE with different α values, ranging from 0.5 to 0.9 with the interval of 0.1, to select test inputs under different sizes, respectively. We then calculated the MSE value for PACE with each α value.

To investigate the impact of dimension reduction on PACE in RQ4, we ran PACE with different dimension numbers (i.e., 2, 4, 8, and 16) of the FastICA algorithm and ran PACE with different dimension reduction algorithms (i.e., the PCA algorithm, the NMF algorithm, and the FA algorithm) to select test inputs under different sizes, respectively. We then calculated the MSE value for PACE with different dimension numbers and different dimension reduction algorithms.

5 RESULTS AND ANALYSIS

5.1 Overall Effectiveness

Tables 2 and 3 present the comparison results among PACE, SRS, CES, and CSS in terms of effectiveness (i.e., the MSE values). Due to the space limit, we separated the comparison results into two tables. In the two tables, we highlighted the best effectiveness among all the compared approaches for each number of selected test inputs using and calculated the average improvement of PACE at all the studied numbers of selected test inputs over each compared approach shown in Column "$\Uparrow_{Im}$." From the two tables, for all the 336 cases (24 subjects $\times$ 14 settings for the number of selected test inputs), PACE performs the best in 87.80% (295 out of 336) cases, while SRS performs the best in 2.08% (7 out of 336) cases, CES performs the best in 6.55% (22 out of 336) cases, and CSS performs the best in 3.57% (12 out of 336) cases, demonstrating the dominant superiority of PACE. We also calculated the average results on all the subjects and found that PACE outperforms all the three compared approaches on average at each studied number of selected test inputs. In particular, the average MSE values of PACE only range from 1.181% to 2.302%, demonstrating that PACE indeed is able to estimate the accuracy of the whole testing set very precisely using a small number of selected test inputs. Moreover, PACE improves the effectiveness by 51.06%, 50.94%, and 70.12% compared with SRS, CES, and CSS on average, respectively. In practice, the accuracy of a DNN model is very important, especially in safety-critical domains. Therefore, the achieved improvements by PACE are indeed important and valuable.

Furthermore, we found that the performance of PACE has fluctuations to some extent on different numbers of selected test inputs. We analyzed the possible reasons for the fluctuations in PACE, which are twofold. First, during the clustering process, it is challenging for PACE to divide all test inputs with different testing capabilities into different groups. That is, there may be noise during the clustering process in PACE. Second, PACE determines the number of test inputs selected from each group according to the proportion of group sizes. However, for the studied numbers of selected test inputs in this work (which is similar to the existing work [55]), it may not always exactly divide each studied number into these groups. That is, there may exist the cases of rounding, which could lead to performance fluctuations especially when the number of selected test inputs is small. Please note that despite such fluctuations in PACE, PACE still outperforms all the compared approaches in most cases (i.e., 87.80% cases). Therefore, PACE is a more reliable choice for DNN test input selection among all these existing approaches. Besides, during the practical usage, when users select the number (that can be exactly divided into each group without rounding) of test inputs, the performance of PACE could become more stable.

Table 2. Effectiveness Comparison Among PACE, SRS, CES, and CSS in Terms of MSE (%) (ID: 1~12)

ID	App.	Number of selected test inputs														$\Uparrow_{Im}$(%)
		50	60	70	80	90	100	110	120	130	140	150	160	170	180	
1	SRS	2.423	2.040	1.679	1.741	1.827	1.699	1.739	1.672	1.595	1.492	1.509	1.465	1.440	1.333	76.94
	CES	2.540	2.160	1.913	1.837	1.880	1.816	1.742	1.613	1.529	1.445	1.326	1.334	1.289	1.262	76.94
	CSS	2.409	2.308	1.925	1.916	1.747	2.222	2.004	1.847	1.694	1.579	1.730	1.652	1.572	1.502	78.89
	PACE	0.742	0.222	0.494	1.110	0.295	0.190	0.315	0.181	0.163	0.574	0.158	0.202	0.434	0.385	—
2	SRS	3.010	2.576	2.438	2.434	2.147	1.978	1.852	1.770	1.680	1.567	1.520	1.591	1.419	1.368	57.30
	CES	2.447	2.027	1.975	1.813	1.757	1.681	1.483	1.446	1.416	1.408	1.330	1.217	1.217	1.216	48.68
	CSS	1.485	1.260	1.145	1.073	1.015	0.944	0.848	0.800	0.717	0.739	0.677	0.631	0.642	0.658	6.84
	PACE	1.249	1.790	1.076	0.494	0.087	0.240	0.483	0.731	0.920	1.082	0.561	0.710	0.871	0.988	—
3	SRS	1.418	1.310	1.258	1.154	1.047	1.043	1.107	1.034	0.954	0.919	0.957	0.896	0.838	0.821	52.13
	CES	1.227	1.120	1.063	0.985	0.911	0.934	0.903	0.902	0.886	0.825	0.839	0.755	0.703	0.709	44.73
	CSS	0.764	0.736	0.627	0.599	0.549	0.548	0.542	0.524	0.526	0.502	0.475	0.447	0.449	0.443	8.74
	PACE	0.603	0.319	0.088	0.070	0.245	0.320	0.419	0.487	0.551	0.611	0.653	0.703	0.732	0.764	—
4	SRS	4.514	4.805	4.491	4.170	3.494	3.157	3.312	3.297	3.219	3.089	2.941	2.987	2.795	2.751	34.94
	CES	3.784	3.821	3.612	3.640	3.496	3.597	3.444	3.426	3.282	3.380	3.433	3.541	3.586	3.535	33.25
	CSS	8.795	8.173	8.202	7.778	7.390	7.128	6.595	6.009	5.688	5.750	5.402	5.168	5.166	4.590	65.64
	PACE	4.470	3.803	3.569	2.748	3.616	2.470	2.288	2.823	1.720	1.899	1.803	1.095	0.470	0.580	—
5	SRS	5.435	4.757	4.425	4.176	3.988	3.983	4.250	3.960	3.603	3.454	3.649	3.610	3.358	3.210	36.72
	CES	5.199	4.702	4.436	4.259	4.576	4.486	4.544	4.580	4.255	4.272	4.353	4.297	4.253	4.397	44.37
	CSS	9.872	9.857	9.074	7.943	7.148	6.644	6.368	6.228	5.917	5.676	5.249	5.102	5.008	4.756	61.42
	PACE	1.161	2.730	2.440	2.730	2.730	3.730	2.730	2.730	2.730	2.016	2.200	2.360	2.262	2.060	—
6	SRS	5.520	5.333	4.674	4.084	3.606	3.578	3.456	3.119	2.965	2.914	2.981	3.097	3.307	3.373	78.67
	CES	4.211	4.230	4.121	3.755	3.633	3.583	3.476	3.274	3.004	3.038	3.111	2.969	3.036	3.067	76.90
	CSS	7.991	7.083	7.307	7.290	6.767	6.683	5.974	5.809	5.422	4.935	4.767	4.658	4.746	4.609	86.95
	PACE	3.140	0.807	0.569	0.390	1.080	1.860	1.217	0.027	0.705	0.293	0.330	0.879	0.316	0.134	—
7	SRS	1.681	1.551	1.456	1.391	1.324	1.187	1.086	1.019	0.994	0.901	0.812	0.762	0.690	0.690	83.71
	CES	1.120	1.019	0.977	0.968	0.973	0.895	0.827	0.836	0.760	0.698	0.708	0.681	0.650	0.636	79.60
	CSS	0.795	0.731	0.666	0.596	0.537	0.552	0.574	0.496	0.484	0.500	0.459	0.408	0.382	0.377	67.61
	PACE	0.080	0.008	0.123	0.051	0.259	0.026	0.232	0.483	0.262	0.309	0.085	0.137	0.044	0.169	—
8	SRS	4.175	3.372	3.098	2.879	2.636	2.337	2.311	2.181	2.168	2.159	2.202	2.093	2.123	2.136	46.02
	CES	4.277	3.779	3.492	2.942	2.894	2.663	2.555	2.510	2.296	2.272	2.274	2.095	2.108	1.968	48.14
	CSS	3.506	3.719	3.229	2.850	2.683	2.471	2.317	2.323	2.203	2.128	2.226	2.215	2.118	2.026	45.56
	PACE	2.668	0.353	0.099	1.143	0.241	1.351	2.261	1.367	1.374	1.379	1.384	0.767	1.976	1.947	—
9	SRS	6.189	5.963	5.672	5.276	4.796	4.560	4.214	3.829	3.794	3.641	3.684	3.399	3.092	3.017	40.37
	CES	6.177	6.053	5.595	5.054	5.063	4.784	4.522	4.244	4.065	3.600	3.358	3.224	3.228	3.118	41.24
	CSS	9.345	8.407	8.075	8.536	8.631	8.601	8.237	7.866	7.301	6.799	6.556	6.418	6.258	6.529	66.55
	PACE	5.051	3.990	3.228	1.050	1.420	2.420	2.051	3.387	1.954	1.916	3.208	1.855	2.596	1.807	—
10	SRS	4.598	4.297	3.344	3.295	3.253	3.019	2.590	2.389	2.406	2.423	2.408	2.402	2.152	2.049	74.47
	CES	5.846	5.858	5.577	5.295	5.159	4.915	5.011	4.783	4.578	4.604	4.643	4.636	4.786	4.726	85.54
	CSS	4.170	3.903	3.767	3.876	3.949	3.677	3.376	3.248	3.126	2.902	2.964	2.895	2.782	2.662	78.26
	PACE	1.578	0.164	1.578	1.639	0.646	0.676	0.070	0.008	0.164	0.396	0.968	0.779	0.612	1.082	—
11	SRS	1.813	1.505	1.411	1.288	1.363	1.267	1.176	1.199	1.152	1.077	1.081	1.026	1.008	0.939	27.79
	CES	1.599	1.344	1.220	1.195	1.101	1.013	1.066	1.082	1.022	0.975	1.011	1.022	0.949	0.879	18.86
	CSS	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
	PACE	1.415	1.389	1.455	1.255	1.382	1.126	1.049	0.717	0.244	0.093	0.400	0.649	0.696	0.954	—
12	SRS	1.647	1.469	1.398	1.293	1.143	1.083	1.012	1.027	0.991	0.999	0.991	0.941	0.831	0.799	50.06
	CES	1.572	1.511	1.311	1.181	1.077	1.037	1.011	0.973	0.869	0.852	0.795	0.747	0.729	0.702	45.50
	CSS	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
	PACE	1.770	1.286	0.521	1.581	0.889	0.350	0.106	0.069	0.123	0.290	0.474	0.333	0.364	0.391	—

*Columns 3–16 present the MSE values (%) for different numbers of selected test inputs. The cell marked with the shading represents the best effectiveness among all the compared approaches in the case. "—" represents the approach cannot be applied in the case. Column "$\Uparrow_{Im}$" represents the average improved rate of the MSE value of PACE at all the studied numbers of selected test inputs over each of the three compared approaches.

Table 3. Effectiveness Comparison Among PACE, SRS, CES, and CSS in Terms of MSE (%) (ID: 13~24)

ID	App.	Number of selected test inputs														$\Uparrow_{Im}$(%)
		50	60	70	80	90	100	110	120	130	140	150	160	170	180	
13	SRS	5.007	4.448	4.099	3.557	3.294	3.115	3.071	2.885	2.949	2.741	2.811	2.701	2.479	2.418	52.04
	CES	4.607	4.321	3.905	3.302	2.845	2.755	2.695	2.470	2.333	2.207	2.309	2.334	2.199	2.200	45.11
	CSS	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
	PACE	6.034	3.015	0.897	2.672	0.663	0.563	0.844	0.277	2.796	2.280	0.943	1.796	0.536	0.269	—
14	SRS	6.587	5.572	4.895	5.400	5.130	4.996	4.882	4.885	4.573	4.238	4.087	3.750	3.560	3.358	80.38
	CES	5.516	5.108	4.912	4.549	4.325	3.960	3.897	3.512	3.392	3.309	3.344	3.205	3.113	3.062	76.92
	CSS	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
	PACE	3.798	0.830	1.970	0.459	1.740	1.578	0.246	0.444	0.153	0.716	0.610	0.492	0.883	0.021	—
15	SRS	3.057	2.799	2.606	2.467	2.330	2.149	2.165	1.975	1.824	1.732	1.740	1.716	1.597	1.518	55.91
	CES	2.580	2.316	2.079	1.917	1.738	1.699	1.616	1.564	1.481	1.404	1.438	1.459	1.407	1.364	44.47
	CSS	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
	PACE	2.479	2.438	1.138	1.695	1.483	1.342	1.611	0.992	0.824	0.496	0.175	0.020	0.010	0.005	—
16	SRS	7.360	7.536	7.023	6.447	5.859	5.733	5.335	5.108	4.743	4.722	4.425	4.280	4.068	3.944	40.47
	CES	5.746	5.238	5.030	4.930	4.940	4.653	4.489	4.335	4.003	3.988	3.852	3.676	3.589	3.300	25.98
	CSS	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
	PACE	6.131	5.832	3.957	5.318	3.996	3.708	2.667	3.177	3.235	2.298	1.000	0.796	2.573	2.665	—
17	SRS	4.438	4.205	3.617	3.451	3.433	2.911	2.578	2.396	2.148	2.176	2.254	2.294	2.165	2.160	61.47
	CES	4.976	4.432	3.993	3.863	3.969	3.593	3.403	3.174	3.203	2.974	2.901	2.854	2.838	2.855	69.08
	CSS	5.450	4.673	4.258	3.821	3.829	3.528	3.246	3.308	3.093	3.088	3.107	3.008	2.893	2.745	69.34
	PACE	0.316	0.284	1.669	1.478	2.428	2.199	1.111	0.203	1.330	0.518	1.801	1.060	0.406	0.175	—
18	SRS	8.067	7.652	6.993	6.876	6.244	5.700	5.179	4.810	4.801	4.423	4.112	4.074	3.832	3.589	45.88
	CES	7.426	6.008	5.914	5.099	4.579	4.727	4.467	4.197	3.961	3.665	3.566	3.402	3.280	3.232	36.09
	CSS	13.815	12.785	11.538	10.262	9.640	8.657	8.055	7.325	7.015	6.567	6.720	6.380	6.177	6.089	65.59
	PACE	1.979	3.079	2.594	1.728	0.115	1.148	3.380	3.545	4.141	3.171	3.033	2.882	3.629	2.608	—
19	SRS	6.452	5.998	6.039	5.549	4.911	4.331	4.229	3.985	3.767	3.886	3.676	3.643	3.387	3.190	23.55
	CES	7.256	6.734	6.345	6.231	6.210	6.008	5.469	5.146	5.073	4.817	4.423	4.198	4.040	3.941	38.55
	CSS	12.653	12.011	11.722	10.714	9.812	8.658	8.540	7.798	7.875	7.418	7.017	6.877	6.409	6.268	61.26
	PACE	0.920	2.080	4.223	3.330	3.147	4.605	7.080	6.247	4.772	2.794	2.413	1.734	2.109	0.726	—
20	SRS	7.279	5.974	5.280	5.393	5.051	4.561	4.374	4.142	3.973	3.628	3.549	3.503	3.508	3.304	57.35
	CES	6.443	5.963	5.668	5.625	5.131	5.212	4.802	4.452	4.686	4.694	4.719	4.591	4.363	4.421	65.07
	CSS	12.256	12.070	11.720	10.181	9.179	7.510	7.347	7.252	7.238	6.568	6.377	5.962	6.080	6.281	76.06
	PACE	2.310	0.357	0.596	1.190	0.357	0.690	0.855	0.477	2.464	3.453	4.310	3.810	1.604	1.310	—
21	SRS	7.919	7.478	6.621	6.417	5.646	5.466	4.890	4.331	4.028	4.198	4.038	4.161	4.086	3.929	36.23
	CES	6.258	5.993	5.327	5.332	5.164	5.120	5.284	5.115	4.985	4.802	4.796	4.819	4.648	4.586	37.67
	CSS	14.096	12.718	11.967	11.351	10.762	9.782	9.638	9.234	8.501	7.995	8.301	8.111	7.742	7.741	66.73
	PACE	1.310	3.870	3.986	4.057	3.005	5.185	4.240	3.429	3.527	2.917	2.341	1.274	2.619	3.042	—
22	SRS	6.262	5.324	5.376	4.792	4.179	4.315	4.526	4.396	4.114	3.794	3.556	3.408	3.300	3.057	62.95
	CES	5.348	5.101	4.520	4.219	4.030	3.870	3.523	3.209	2.976	2.805	2.656	2.599	2.522	2.539	53.27
	CSS	11.089	10.264	9.075	9.012	8.730	8.548	7.897	8.092	7.604	7.004	6.476	6.421	6.717	6.355	80.39
	PACE	0.667	1.716	1.315	2.093	1.364	0.185	0.816	1.161	1.937	2.838	2.786	0.933	1.747	1.399	—
23	SRS	7.231	6.515	6.084	5.568	5.835	5.538	5.176	5.070	4.627	4.430	4.216	4.023	3.855	3.858	43.28
	CES	6.581	5.555	4.999	4.949	4.867	4.629	4.475	4.111	3.848	3.666	3.524	3.340	3.241	3.021	32.08
	CSS	9.709	9.646	9.300	9.108	8.788	7.959	7.247	6.865	6.311	6.126	5.980	6.325	6.053	5.735	61.37
	PACE	3.980	1.599	3.792	4.737	3.003	0.647	0.430	3.750	2.594	2.988	2.682	2.412	2.948	3.785	—

(Continued)

Table 3. Continued

ID	App.	Number of selected test inputs														$\Uparrow_{Im}$(%)
		50	60	70	80	90	100	110	120	130	140	150	160	170	180	
24	SRS	3.043	2.584	2.806	2.837	2.680	2.473	2.504	2.244	2.122	1.892	1.729	1.674	1.686	1.642	43.07
	CES	3.068	3.102	3.016	2.893	2.693	2.686	2.690	2.679	2.620	2.574	2.452	2.483	2.512	2.453	52.49
	CSS	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	PACE	1.389	2.192	2.613	1.812	0.976	1.587	1.801	0.298	0.890	0.506	0.865	1.040	1.304	1.075	–
Avg	SRS	4.797	4.377	4.033	3.831	3.551	3.341	3.209	3.030	2.883	2.771	2.705	2.646	2.524	2.435	51.06
	CES	4.408	4.062	3.792	3.576	3.459	3.347	3.225	3.068	2.939	2.845	2.798	2.728	2.679	2.633	50.94
	CSS	7.541	7.079	6.682	6.289	5.950	5.536	5.224	5.001	4.748	4.487	4.381	4.275	4.188	4.080	70.12
	PACE	2.302	1.840	1.833	1.868	1.465	1.591	1.596	1.542	1.649	1.493	1.466	1.197	1.323	1.181	–

*Row "Avg" represents the average results on all the subjects. Please note that CSS cannot be applied to regression models and multi-label classification models, and thus the average results for CSS are based on all the classification models except DeepSpeech and the average improvement of PACE over CSS is also based on all the classification models except DeepSpeech.

Table 4. Statistical Analysis of the Effectiveness of PACE Compared with SRS, CES, and CSS

PACE v.s.	50	60	70	80	90	100	110	120	130	140	150	160	170	180
SRS	**0.828**	**0.835**	**0.826**	**0.814**	**0.839**	**0.800**	**0.806**	**0.788**	**0.752**	**0.774**	**0.776**	**0.828**	**0.786**	**0.806**
CES	**0.788**	**0.821**	**0.813**	**0.783**	**0.835**	**0.799**	**0.804**	**0.786**	**0.752**	**0.792**	**0.786**	**0.819**	**0.795**	**0.807**
CSS	**0.855**	**0.869**	**0.841**	**0.824**	**0.862**	**0.837**	**0.820**	**0.806**	**0.789**	**0.792**	**0.785**	**0.803**	**0.803**	**0.817**

*Each cell represents the $\hat{A}_{12}$ value and the bold value indicates that PACE indeed significantly outperforms the compared approach. The statistical analysis for PACE v.s. SRS and PACE vs. CES is conducted on all the subjects while the statistical analysis for PACE vs. CSS is conducted on all the classification models except DeepSpeech, since CSS cannot be applied to regression models and multi-label classification models.

Please note that, as presented in Section 3.1, PACE does not rely on the classes, but instead relies on the input features of test inputs, and thus the performance of PACE is not affected by the number of classes. Moreover, same as the existing work [55], PACE aims to estimate the accuracy of the whole testing set rather than the accuracy for each class. Therefore, PACE is able to perform well regardless of the number of classes (e.g., 10 classes for CIFAR-10 or 1,000 classes for ImageNet).

Statistical analysis. To further confirm our observations, we performed *a paired sample Wilcoxon signed-rank test* [100] at the significance level of 0.05 to investigate whether PACE can significantly outperform each compared approach across all the subjects, and then performed *the Vargha-Delaney effect size measure* [96] to investigate the difference degree between PACE and each compared approach. Following the existing work [65], the difference degree is characterized as small, medium, and large when the effect size $\hat{A}_{12}$ is larger than 0.56, 0.64, and 0.71, respectively. Table 4 shows the results of statistical analysis. From this table, we found that all the values are bold, indicating that PACE is able to significantly outperform all the compared approaches at each studied number of selected test inputs. Moreover, all the effect size values are larger than 0.71, demonstrating that the differences between PACE and all the compared approaches are large at each studied number of selected test inputs.

Effectiveness on classification models and regression models. We considered the models conducting different tasks, including regression (ID: 11~16) and classification, to evaluate the effectiveness of PACE in our study. Table 5 shows the average results of PACE on classification models and regression models, respectively. From this table, we find that on average, PACE outperforms all the three compared approaches for classification models at each studied number of

Table 5. Effectiveness Comparison Among PACE, SRS, CES, and CSS in Terms of the Average MSE Values (%) for Classification Models and Regression Models

Task	App.	Number of selected test inputs														$\Uparrow_{Im}$(%)
		50	60	70	80	90	100	110	120	130	140	150	160	170	180	
C	SRS	4.981	4.541	4.186	3.971	3.672	3.435	3.298	3.091	2.942	2.833	2.766	2.727	2.613	2.526	49.95
	CES	4.677	4.314	4.030	3.820	3.721	3.622	3.479	3.316	3.190	3.086	3.023	2.946	2.906	2.871	52.26
	CSS	7.541	7.079	6.682	6.289	5.950	5.536	5.224	5.001	4.748	4.487	4.381	4.275	4.188	4.080	68.54
	PACE	1.867	1.631	1.892	1.769	1.390	1.640	1.765	1.741	1.789	1.648	1.755	1.368	1.482	1.335	—
R	SRS	4.245	3.888	3.572	3.409	3.186	3.057	2.940	2.847	2.705	2.585	2.522	2.402	2.257	2.162	54.53
	CES	3.603	3.306	3.076	2.846	2.671	2.520	2.462	2.323	2.183	2.123	2.125	2.074	1.998	1.918	46.03
	CSS	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
	PACE	3.604	2.465	1.656	2.163	1.692	1.445	1.087	0.946	1.229	1.029	0.600	0.681	0.844	0.717	—

*Rows "C" and "R" represent the average results on all the classification models and regression models, respectively. Please note that CSS cannot be applied to multi-label classification models, and thus the average results of CSS for classification models do not include DeepSpeech and the average improvement of PACE over CSS for classification models also do not include DeepSpeech.

Table 6. Effectiveness Comparison among PACE, SRS, CES, and CSS in Terms of the Average MSE Values (%) for High-accuracy Models and Low-accuracy Models

Acc.	App.	Number of selected test inputs														$\Uparrow_{Im}$(%)
		50	60	70	80	90	100	110	120	130	140	150	160	170	180	
High	SRS	2.846	2.519	2.283	2.203	2.107	1.922	1.829	1.719	1.640	1.576	1.564	1.533	1.450	1.405	57.82
	CES	2.841	2.606	2.420	2.263	2.196	2.085	2.028	1.960	1.878	1.821	1.792	1.753	1.744	1.706	61.76
	CSS	2.654	2.476	2.231	2.104	2.044	1.992	1.844	1.792	1.692	1.634	1.662	1.608	1.548	1.488	58.34
	PACE	1.299	0.950	0.987	1.121	0.812	0.855	0.860	0.503	0.622	0.569	0.684	0.582	0.677	0.721	—
Low	SRS	6.448	5.950	5.513	5.208	4.772	4.541	4.376	4.140	3.935	3.782	3.671	3.587	3.433	3.308	48.66
	CES	5.735	5.294	4.953	4.688	4.528	4.414	4.237	4.006	3.836	3.711	3.650	3.553	3.469	3.417	46.46
	CSS	10.962	10.301	9.798	9.218	8.685	8.017	7.590	7.248	6.887	6.484	6.284	6.142	6.036	5.895	70.60
	PACE	3.150	2.593	2.549	2.500	2.018	2.214	2.219	2.421	2.518	2.275	2.128	1.717	1.869	1.570	—

*Rows "High" and "Low" represent the average results on all the high-accuracy models and low-accuracy models, respectively. Since CSS cannot be applied to regression models and multi-label classification models, the results of CSS and the improvement of PACE over CSS are based on all the corresponding classification models except DeepSpeech.

selected test inputs. The average MSE values of PACE only range from 1.335% to 1.892%, improving the effectiveness by 49.95%, 52.26%, and 68.54% compared with SRS, CES, and CSS, respectively. For regression models, PACE also outperforms all the three compared approaches at almost all the studied numbers of selected test inputs (except 50) on average. For 50, the average MSE value of PACE (i.e., 3.604%) is just slightly larger than the best effectiveness (i.e., 3.603%). Also, the average improvements of PACE compared with SRS and CES achieve 54.53% and 46.03%, respectively. These results demonstrate that, regardless of classification models and regression models, PACE has extremely small deviation for estimating the accuracy of the whole testing set using a small number of selected test inputs.

Effectiveness on high-accuracy models and low-accuracy models. We then analyzed the effectiveness of PACE for the models with different accuracy, since in practice the models with various accuracy may exist. Table 6 shows the average results of PACE on high-accuracy models and low-accuracy models. From this table, we found that PACE is able to outperform all the compared approaches at each studied number of selected test inputs on average for both high-accuracy models and low-accuracy models, achieving at least 46.46% improvements. More specifically, for

Table 7. Effectiveness Comparison Among PACE, SRS, CES, and CSS in Terms of the Average MSE Values (%) for Different Types of Test Inputs

Type	App.	Number of selected test inputs														$\Uparrow_{Im}$(%)
		50	60	70	80	90	100	110	120	130	140	150	160	170	180	
Ori.	SRS	3.687	3.285	3.049	2.877	2.743	2.57	2.452	2.325	2.211	2.113	2.071	1.998	1.892	1.836	54.05
	CES	3.598	3.312	3.05	2.861	2.798	2.655	2.555	2.428	2.313	2.208	2.14	2.076	2.059	2.007	55.60
	CSS	4.872	4.565	4.207	4.139	4.046	3.905	3.629	3.537	3.306	3.137	3.065	3.042	2.986	2.903	67.04
	PACE	1.654	1.178	1.389	1.424	1.018	0.870	0.857	0.988	0.962	1.039	1.233	0.891	1.133	1.148	–
Mut.	SRS	5.503	5.089	4.656	4.468	4.153	3.998	3.863	3.713	3.522	3.358	3.266	3.112	2.926	2.809	56.99
	CES	4.612	4.246	3.981	3.674	3.462	3.267	3.174	2.970	2.802	2.727	2.736	2.668	2.577	2.481	48.29
	CSS	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	PACE	4.610	3.029	1.990	2.536	1.971	1.798	1.342	1.222	1.752	1.448	0.682	0.776	1.000	0.740	–
Gen.	SRS	7.429	6.775	6.233	6.059	5.463	5.014	4.668	4.317	4.142	4.034	3.844	3.845	3.703	3.503	41.00
	CES	6.846	6.174	5.813	5.571	5.271	5.267	5.006	4.728	4.676	4.495	4.376	4.253	4.083	4.045	44.52
	CSS	13.205	12.396	11.737	10.627	9.848	8.652	8.395	7.902	7.657	7.137	7.104	6.832	6.602	6.595	67.38
	PACE	1.630	2.346	2.850	2.576	1.656	2.907	3.889	3.424	3.726	3.084	3.024	2.425	2.490	1.921	–

*Rows "Ori.," "Mut.," and "Gen." represent the average results on all the original test inputs, mutated test inputs, and automatically generated test inputs, respectively. Since CSS cannot be applied to regression models and multi-label classification models, the results of CSS and the improvement of PACE over CSS are based on all the corresponding classification models except DeepSpeech. In particular, since the models with mutated test inputs are all regression models in our study, CSS cannot be applied to them.

the low-accuracy models produced via mutation (ID: 4 ∼ 6), the existing work has demonstrated that CES performs better than CSS and SRS, and our study further demonstrated that PACE outperforms the state-of-the-art CES as shown in Table 2. Moreover, from Table 2, for the real-world low-accuracy model (ID: 9, 22, and 23), PACE performs the best among all the approaches for each studied number of selected test inputs. That is, PACE is able to outperform the other three approaches for both mutated low-accuracy models and real-world low-accuracy models.

Effectiveness on different types of test inputs. We further analyzed the effectiveness of PACE for different types of test inputs (including original test inputs, mutated test inputs, and automatically generated test inputs), whose results are shown in Table 7. From this table, we found that on average, PACE performs the best among all the approaches at each studied number of selected test inputs for various types of test inputs, and all the average improvements of PACE over the other approaches are larger than 41.00%. More specifically, from Table 3, for the mutated test inputs (ID: 13∼16), we first confirmed the conclusion from the existing work [55], i.e., CES outperforms SRS in this scenario. Also, we found that PACE performs better than both CES and SRS in most cases, demonstrating the effectiveness of PACE for the mutated test inputs. For the automatically generated test inputs (ID: 18 ∼ 21), our work is the first one to investigate the effectiveness of various selection approaches in this scenario. We found that PACE also outperforms the other three approaches, and surprisingly, the baseline SRS performs better than the state-of-the-art CES and CSS for the automatically generated test inputs. To sum up, PACE is able to stably perform the best for various types of test inputs.

Efficiency comparison. It is also interesting to investigate the efficiency of these approaches. We reported the time cost spent on selection for each approach on each subject in Table 8, where we recorded the time cost spent on selecting 180 test inputs as the representative. From this table, although the average, minimum, and maximum time costs spent on selecting 180 test inputs on all the subjects for PACE are larger than those for the other three compared approaches, PACE

Table 8. Efficiency Comparison Among PACE, SRS, CES, and CSS in Terms of the Time Cost Spent on Selecting 180 Test Inputs (Minutes)

ID	SRS	CES	CSS	PACE
1	2.667E-06	0.043	0.040	6.518
2	9.200E-06	0.030	0.040	3.238
3	1.667E-06	0.021	0.034	1.267
4	2.333E-06	0.028	0.034	1.202
5	4.633E-06	0.028	0.034	1.092
6	3.667E-06	0.028	0.034	1.197
7	2.983E-06	0.340	0.596	0.547
8	3.733E-06	0.421	0.794	1.437
9	3.917E-06	1.479	2.761	3.335
10	1.667E-05	0.068	0.101	9.650
11	3.667E-06	0.052	—	0.153
12	3.333E-06	0.099	—	0.452
13	2.767E-06	0.052	—	0.227
14	2.767E-06	0.096	—	0.317
15	3.100E-06	0.064	—	0.142
16	2.983E-06	0.033	—	0.583
17	1.717E-06	0.029	0.027	3.342
18	2.750E-06	0.033	0.034	4.157
19	3.317E-06	0.420	0.808	3.155
20	2.867E-06	0.040	0.037	1.237
21	1.750E-06	0.036	0.028	1.542
22	2.026E-05	36.372	29.327	46.138
23	1.965E-05	22.552	20.945	26.587
24	1.931E-05	2.594	—	16.395
Avg.	5.905E-06	2.707	3.275	5.579
Min.	1.667E-06	0.021	0.027	0.142
Max.	2.026E-05	36.372	29.327	46.138
Std.	6.038E-06	8.329	8.137	10.339

just spent several minutes, which is acceptable in practice. In particular, the most costly part of PACE is clustering, but it is performed only once in the beginning and then the selection part is incrementally performed based on the clustering result. That is, the time cost of PACE does not largely increase when the required number of selected test inputs increases. By taking subject 24 (DeepSpeech based on Speech-Commands) as an example, when the number of selected test inputs is 180, the time cost of PACE is 16.395 min while when the number of selected test inputs is 1,000, the time cost of PACE is 17.363 min, which further confirms the above hypothesis.

In summary, PACE is able to precisely estimate the accuracy of the whole testing set using a small number of selected test inputs with only 1.181%∼2.302% deviations on average, significantly outperforming all the three compared approaches. In particular, PACE achieves great effectiveness for various types of models and various types of test inputs, fitting complex scenarios in practice very well. Therefore, PACE is indeed a practical approach with the best effectiveness and acceptable efficiency.

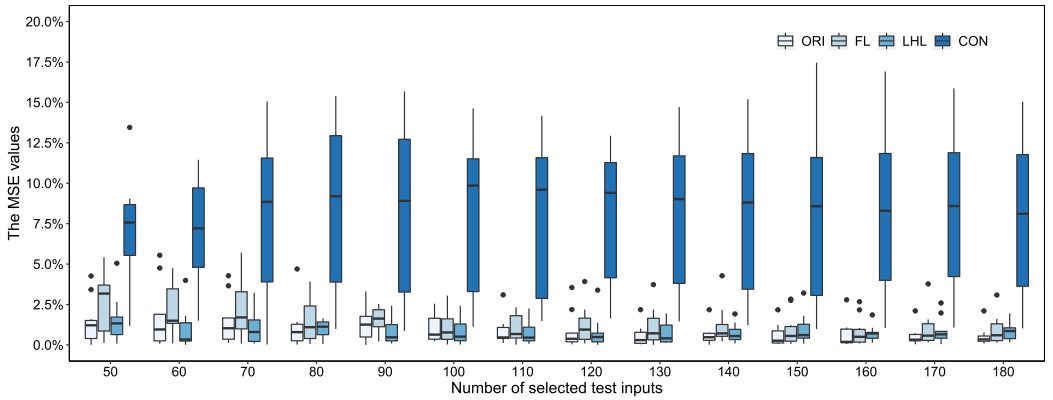


Fig. 4. The MSE values of PACE with different features.

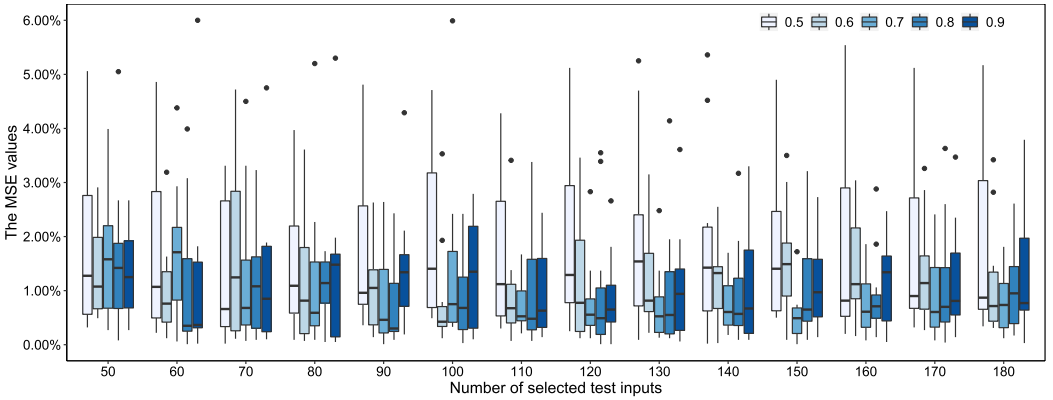


Fig. 5. The MSE values of PACE with different α values.

5.2 Impact of Different Features

We investigated the impact of different features on PACE, whose results are shown in Figure 4. In this figure, the x-axis represents the different numbers of selected test inputs and the y-axis represents the MSE values. The box plots show the median and interquartile ranges of the MSE values on all the subjects. From this figure, we found that the CON feature performs the worst among the four types of features, because CON is actually the prediction result for a test input and cannot reflect the testing capability of the test input well. For the other three types of features, ORI and our default setting LHL perform better than FL. That demonstrates that LHL indeed is a good choice for the default setting of PACE. In the meanwhile, the most basic feature ORI performs surprisingly good, indicating that such basic feature is enough to reflect the testing capabilities of test inputs well. In particular, ORI is a kind of static features, and thus it provides an opportunity to make PACE more efficient by avoiding running the DNN models.

5.3 Impact of Threshold α

We investigated the impact of different values of the threshold α, whose results are shown in Figure 5. From this figure, we found that the α values setting of 0.5 performs the worst among all the α values. This is because the size of the minority space tends to be smaller than that of the grouped test inputs actually, and thus this setting leads to poor performance. Moreover, we found

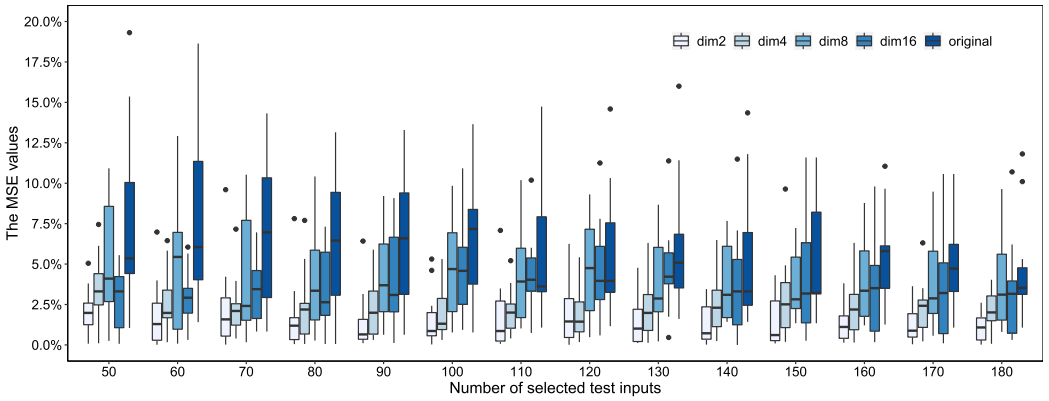


Fig. 6. The MSE values of PACE with different dimension numbers of FastICA.

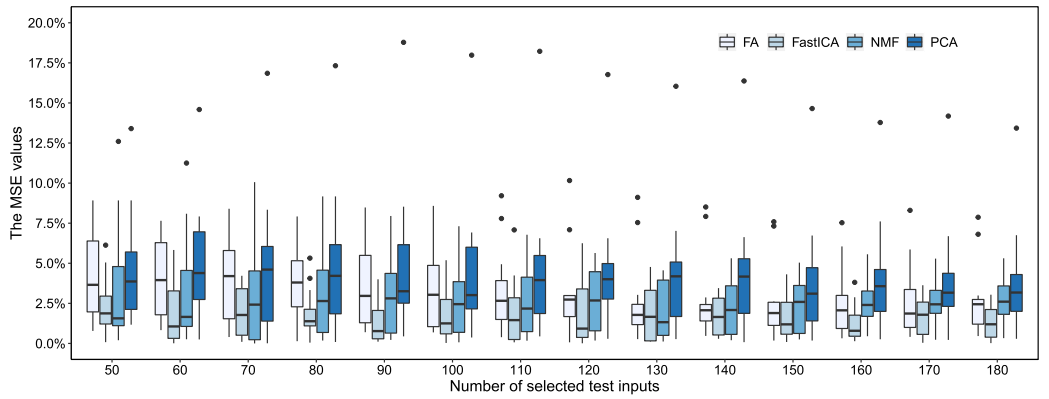


Fig. 7. The MSE values of PACE with different dimension reduction algorithms.

that the α values of 0.7 and 0.8 perform relatively better than those of 0.6 and 0.9, indicating the effectiveness of the default threshold α in PACE. Also, we found that for different subjects, the best α values may be different, and thus in the future we plan to propose to set the α value dynamically by considering the characteristics of the used subject.

5.4 Impact of Dimension Reduction

We investigated the impact of different dimension numbers of the FastICA algorithm on PACE, whose results are shown in Figure 6. Here, we studied four different dimension numbers and the case without dimension reduction (represented by "original" in Figure 6). From this figure, we found that PACE without dimension reduction performs the worst, demonstrating that dimension reduction indeed improves the effectiveness of PACE. Also, with the dimension number decreasing, the effectiveness of PACE becomes better in general. The reason is that the performance of HDBSCAN can be affected by the dimension of features. When the dimension of features is high, its performance could be largely decreased [66]. Therefore, according to Figure 6, setting the dimension number to be 2 is a reasonable choice, and achieves the best effectiveness among all the studied dimension numbers in general.

We then investigated the impact of different dimension reduction algorithms on PACE, whose results are shown in Figure 7. From this figure, we found that FastICA performs the best among

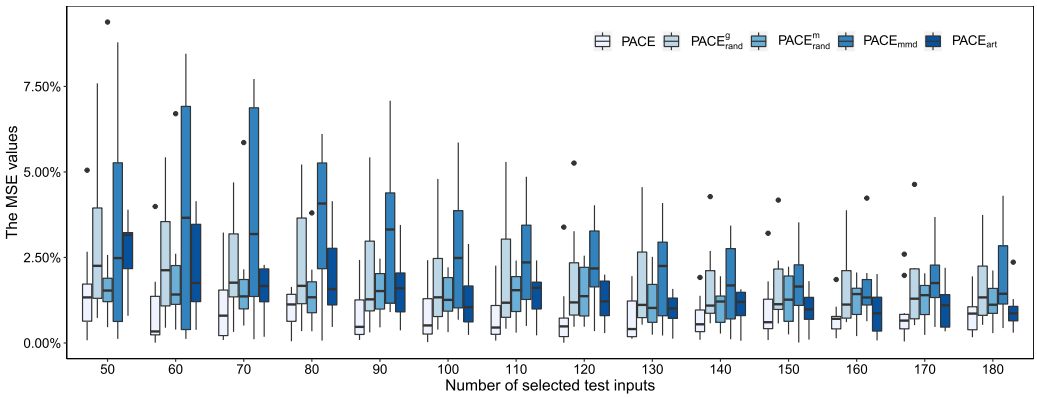


Fig. 8. Comparison results between PACE and its variants in terms of the MSE values.

the four studied dimension reduction algorithms, demonstrating that FastICA is indeed a good choice for dimension reduction in PACE. This is because FastICA aims to find independent components and is able to find underlying factors, while the other three dimension reduction algorithms fail to find those underlying factors. Therefore, FastICA performs better than them. Actually, our approach PACE is not specific to FastICA. For example, in Figure 7, NMF achieves similar effectiveness to FastICA when the number of selected test inputs is small. Therefore, it is also possible to use other effective dimension reduction algorithms to further improve the effectiveness of PACE. Currently, we used FastICA as the default dimension reduction algorithm in PACE due to its effectiveness.

5.5 Contribution of Each Component in PACE

PACE consists of three main components: clustering, MMD-critic-based selection, and adaptive random selection. We investigated the contribution of each component in PACE, whose results are shown in Figure 8. From this figure, first of all, we found that PACE performs better than all the four variants of PACE, demonstrating the contributions of all the components. By comparing PACE$^{g}_{rand}$ and PACE$^{m}_{rand}$ the former performs worse than the latter, indicating that the MMD-critic-based selection for groups is more important than the adaptive random selection for the minority space. This is as expected, because the size of the grouped test inputs is larger than that of the minority space, and thus the selection method for the former makes more contributions. In addition, by comparing PACE$_{mmd}$ and PACE$_{art}$, the former performs worse than the latter, indicating that without clustering the adaptive random selection method is able to achieve better effectiveness than the MMD-critic-based selection method for the whole testing set. This is because without clustering, all the test inputs with different testing capabilities are mixed together, and the adaptive random selection method is more likely to select the test inputs with various testing capabilities by selecting the test inputs with the largest distances.

6 DISCUSSION

6.1 Extensions of PACE

Our experiments have demonstrated that PACE achieves great effectiveness for estimating the accuracy of the whole testing set by selecting a small number of test inputs. There are some possible directions to further improve PACE.

First, as demonstrated by Section 5.2, the static feature ORI is able to achieve the similar effectiveness with our default setting of LHL. Each of them can perform better than the other in some

cases. Therefore, it is intuitive that combining them may produce better effectiveness. In particular, we made the first attempt to combine LHL and ORI by simply concatenating the two feature vectors for each test input, and then conducted a preliminary study using the two DNN models (i.e., VGG-16 and ResNet-20) with the testing set CIFAR-10. The experimental results showed that the average deviation achieved by PACE with the combined features on the two models at all the studied numbers of selected test inputs is 0.299%, improving PACE with the default LHL feature by 13.80%. That demonstrated that combining them is indeed a promising direction to improve PACE. In the future, we will explore more effective methods to combine different types of features to further improve PACE.

Second, in PACE the HDBSCAN algorithm is adopted to perform clustering due to several reasons presented in Section 3.2. However, the efficiency of PACE performs worse than the compared approaches due to the cost of clustering. In the future, we may explore other clustering algorithms with very low overhead to make PACE more practical. In addition, it is possible for the HDBSCAN algorithm to produce the extreme case that almost all the test inputs are not clustered into any group due to its noise-assignment-supporting mechanism [66]. Such poor performance of clustering may lead to poor performance of PACE. However, such extreme case is rare, even does not occur in our study for all the used 24 pairs of DNN models and testing sets. In case such extreme case occurs, we can try to relieve it by tuning the parameters of this algorithm or using other effective clustering algorithms.

Third, the current goal of PACE is to precisely estimate the accuracy of the whole testing set, which is a single-objective problem. In practice, there are several objectives that are required to be satisfied, e.g., keeping the same coverage with the whole testing set. In the future, we plan to extend PACE to solve the problem of test input selection with multiple objectives.

Fourth, in our study we evaluated the performance of PACE in the domains of normal image classification, autonomous driving, and speech-to-text engine, but our approach could be generalized to more other domains such as natural language processing. This is because PACE only relies on input features, e.g., the output of the last hidden layer in a DNN (LHL features), and it is feasible for all the inputs to extract these features. In the future, we will further evaluate the performance of PACE in various domains.

6.2 Threats to Validity

The *internal* threat to validity mainly lies in the implementations of PACE and the compared approaches, and the experimental scripts in our study. To reduce this threat, we adopted the implementations of the compared approaches released by the authors, implemented PACE based on some existing libraries (presented in Section 4.4), and carefully checked the code of our approach PACE and experimental scripts.

The *external* threats to validity mainly lie in the DNN models under test and the testing sets used in our study. Regarding the used DNN models, we adopted the models trained based on popular datasets (including MNIST, CIFAR-10, CIFAR-100, SVHN, Driving, Fashion-MNIST, ImageNet, and Speech-Commands). To reduce the threat from the DNN models, we considered different types of models from several aspects, including classification models and regression models, high-accuracy models and low-accuracy models (real-world low-accuracy models and mutated low-accuracy models), and CNN models and a RNN model (i.e., DeepSpeech). Regarding the used testing sets, we also considered different types of test inputs, including the original test inputs, the mutated test inputs, and the automatically generated test inputs. To further reduce these threats, we will apply PACE to more DNN models and testing sets with great diversity in the future.

The *construct* threats to validity main lie in the parameters in PACE, randomness, the measurements used in our study, the studied numbers of selected test inputs, the used method for

Table 9. Effectiveness Comparison Among PACE, SRS, CES, and CSS in Terms of Estimating Top-5 Accuracy by Taking Subjects 22 and 23 as Examples

ID	App.	Number of selected test inputs														$\Uparrow_{Im}$(%)
		50	60	70	80	90	100	110	120	130	140	150	160	170	180	
22	SRS	5.088	4.890	4.730	4.602	4.394	3.863	3.307	3.222	3.164	2.948	3.008	2.883	2.725	2.676	48.29
	CES	4.608	4.288	4.266	4.185	4.112	4.100	4.137	3.890	3.893	3.841	3.766	3.582	3.582	3.411	49.57
	CSS	7.141	6.382	5.681	5.948	5.947	5.353	5.094	5.006	4.720	4.345	4.064	4.102	3.900	3.639	62.17
	PACE	4.632	5.060	3.464	3.464	1.724	1.337	2.689	1.386	0.375	0.747	1.169	0.942	0.655	0.936	—
23	SRS	4.384	4.087	3.602	3.458	3.408	3.077	2.873	2.873	2.707	2.513	2.546	2.388	2.496	2.487	29.28
	CES	4.128	3.849	3.823	3.621	3.477	3.271	2.945	3.029	3.049	2.999	2.673	2.477	2.436	2.396	30.98
	CSS	6.347	5.863	5.322	4.704	4.300	3.603	3.488	3.358	3.342	2.974	2.864	3.058	3.068	3.046	45.40
	PACE	4.449	5.531	3.697	3.357	2.118	2.068	2.027	1.915	1.891	1.870	0.553	0.616	1.187	1.209	—

automatically generating adversarial inputs, and the studied feature type. PACE involves several parameters, such as the threshold α and the clustering parameters. Regarding the threshold α, we conducted a study to evaluate the impact of different α values, and the results demonstrate that the default setting of α (i.e., 0.8) is a good choice (presented in Section 5.3). Regarding the clustering parameters, we set them based on a small dataset and used the same parameters for all the subjects. Also, our used clustering algorithm HDBSCAN is demonstrated to be robust to parameter selection. We presented the specific settings of the parameters in Section 4.4. Moreover, in Section 4.4 we also presented the conditions of applying dimension reduction. In the future, we will further investigate the impact of these parameters. To reduce the threat from randomness involved in our study (including the three compared approaches and three variants PACE^{g}_{rand}, PACE^{m}_{rand}, PACE_{art}), we repeated each of them 50 times and calculated the effectiveness using Equation (6). Regarding the measurements used in our study, we used both *Mean-squared Errors* and the time cost spent on selection to measure the effectiveness and efficiency of each approach, respectively. To reduce this threat, in the future we will use more metrics to measure the effectiveness of each approach more sufficiently, such as measuring the interpretability by communicating with the persons that are responsible to label test inputs. In particular, the accuracy estimated by both our work and the existing work [55] refers to Top-1 accuracy, and it is actually also interesting to investigate the effectiveness of PACE for estimating Top-n accuracy. Here, we conducted an experiment by taking subjects 22 and 23 and Top-5 accuracy as examples to investigate it, whose results are shown in Table 9. From this table, in terms of estimating Top-5 accuracy, for subject 22 PACE improves the effectiveness by 48.29%, 49.57%, and 62.17% compared with SRS, CES, and CSS on average, respectively, while for subject 23 PACE improves the effectiveness by 29.28%, 30.98%, and 45.40% compared with SRS, CES, and CSS on average, respectively. The results demonstrate that, regardless of Top-1 accuracy or Top-5 accuracy, PACE is able to significantly outperform all the three compared approaches.

Regarding the studied numbers of selected test inputs, we studied 14 different numbers of selected test inputs similar to the existing work [55] in our study, which range from 50 to 180 with the interval of 10. However, the studied numbers may not represent other numbers of selected test inputs. To reduce this threat, we further evaluated the effectiveness of PACE using larger numbers of selected test inputs. Here, we take subjects 22 and 23 as examples and the number of selected test inputs ranges from 100 to 1,000 with the interval of 100, whose results are shown in Table 10. From this table, when the number of selected test inputs ranges from 100 to 1,000, PACE always performs better than all the three compared approaches for both subjects 22 and 23. On average, PACE improves the effectiveness by 68.33%, 64.79%, and 90.31% compared with SRS, CES, and CSS,

Table 10. Effectiveness Comparison Among PACE, SRS, CES, and CSS in Terms of the Average MSE Values (%) When the Selected Number Ranges from 100 to 1,000 by Taking ImageNet as an Example

ID	App.	Number of selected test inputs										$\Uparrow_{Im}$(%)
		100	200	300	400	500	600	700	800	900	1,000	
22	SRS	4.315	2.730	1.569	1.625	1.586	1.511	1.429	1.415	1.224	1.352	68.33
	CES	4.629	2.478	2.478	1.452	0.631	0.851	0.930	0.827	0.951	1.369	64.79
	CSS	8.548	7.395	7.103	6.395	5.070	3.478	3.556	4.083	3.687	3.595	90.31
	PACE	0.185	1.124	1.495	1.139	0.200	0.289	0.556	0.024	0.103	0.340	–
23	SRS	5.538	3.802	3.417	1.734	2.036	1.382	1.691	1.896	1.901	1.397	68.08
	CES	4.629	2.548	3.282	2.414	2.975	3.019	2.789	2.152	2.064	1.985	73.67
	CSS	6.126	4.411	5.375	3.508	1.473	1.415	2.570	1.692	1.730	1.353	69.68
	PACE	0.647	1.102	0.599	1.26	1.466	0.067	0.020	0.016	0.488	1.166	–

Table 11. Effectiveness Comparison Among PACE, SRS, CES, and CSS in Terms of the Average MSE Values (%) When Using FGSM and C&W by Taking CIFAR-10 and ResNet-20 as an Example

Adv.	App.	Number of selected test inputs														$\Uparrow_{Im}$(%)
		50	60	70	80	90	100	110	120	130	140	150	160	170	180	
FGSM	SRS	6.362	5.300	5.000	4.508	4.853	4.285	3.972	4.338	4.327	4.118	3.660	3.588	3.732	3.498	48.94
	CES	6.656	6.122	5.432	5.292	5.273	4.866	4.956	5.049	4.891	4.695	4.540	4.560	4.409	4.191	54.97
	CSS	12.193	11.577	10.722	10.330	10.062	9.147	8.603	8.709	8.116	8.338	7.703	7.421	7.552	7.844	75.89
	PACE	5.585	7.840	6.220	4.090	2.840	1.426	0.113	0.493	0.532	0.697	0.493	1.259	1.278	1.604	–
C&W	SRS	7.533	6.722	5.748	5.498	5.083	4.834	5.118	5.076	4.677	4.647	4.685	4.448	4.250	4.353	66.66
	CES	6.881	6.425	5.804	5.440	5.028	4.641	4.696	4.923	4.423	4.224	3.984	3.784	3.537	3.350	63.43
	CSS	12.953	11.762	11.165	9.922	9.171	8.813	8.486	7.431	7.455	7.148	6.067	5.664	5.129	5.100	78.46
	PACE	7.160	3.827	1.446	2.160	1.604	1.160	0.113	1.173	0.237	1.411	1.507	1.590	1.664	1.173	–

respectively, for subject 22, while improves the effectiveness by 68.08%, 73.67%, and 69.68% compared with SRS, CES, and CSS, respectively, for subject 23. The results demonstrate that PACE also performs the best among all these approaches when the number of selected test inputs is large.

Regarding the used method for automatically generating adversarial inputs, we have evaluated the effectiveness of PACE on the testing sets including adversarial inputs generated via Basic Iterative Method (BIM) in our study (ID: 18-21). However, this adversarial input generation method may not represent other adversarial input generation methods. To reduce this threat, we further investigated the effectiveness of PACE on the testing sets including adversarial test inputs generated via the other two widely used adversarial input generation methods, i.e., FGSM [33] and C&W [10], by taking CIFAR-10 and ResNet-20 as an example following the process described in Section 4.1. Table 11 shows the effectiveness of PACE on the testing sets including adversarial test inputs generated via the two adversarial input generation methods (i.e., FGSM and C&W). From this table, we found that PACE still performs the best among all the compared approaches on average when using both FGSM and C&W, and all the average improvements of PACE over the other approaches are larger than 48.94%, demonstrating the effectiveness of PACE when using different adversarial input generation methods.

Regarding the studied feature type, we used input features in PACE rather than coverage features as explained in Section 3.1. Here, we conducted an experiment to further investigate whether coverage features can perform better than input features. In particular, besides CSS and CES, Li et al. [55] also preliminarily studied the effectiveness of coverage features (i.e., surprise coverage [45], the state-of-the-art coverage) used in a similar way to CSS. We call the method of surprise coverage *CSS-COV*. Following this existing work [55], we also compared PACE with CSS-COV by

Table 12. Effectiveness Comparison Between PACE and CSS-COV in Terms of the Average MSE Values (%) by Taking the Subjects Whose ID Is 3 and 4 as Example

ID	App.	Number of selected test inputs														$\Uparrow_{Im}$(%)
		50	60	70	80	90	100	110	120	130	140	150	160	170	180	
3	CSS-COV	3.063	2.586	2.256	2.359	2.353	2.125	1.996	1.840	1.714	1.611	1.527	1.455	1.388	1.336	72.65
	PACE	0.603	0.319	0.088	0.070	0.245	0.320	0.419	0.487	0.551	0.611	0.653	0.703	0.732	0.764	–
4	CSS-COV	8.437	7.617	7.033	6.377	6.025	5.973	5.877	5.749	5.480	5.244	4.759	4.785	4.747	4.842	61.68
	PACE	4.470	3.803	3.569	2.748	3.616	2.470	2.288	2.823	1.720	1.899	1.803	1.095	0.470	0.580	–

taking the subjects whose ID is 3 and 4 as example following the existing work [55], whose results are shown in Table 12. From this table, we found that PACE always performs better than CSS-COV at each studied number of selected test inputs for both of the subjects, and the average improvements of PACE over CSS-COV are 72.65% and 61.68% for the two subjects, respectively. The results demonstrate that the effectiveness of coverage features is not good to some degree, which is consistent with the conclusion from the existing work [55].

7 RELATED WORK

7.1 Deep Neural Network Testing

Besides improving the efficiency of DNN testing, there are a large number of studies focusing on proposing various metrics to measure test adequacy for DNN and designing various approaches to generate adversarial inputs in the literature [26, 40, 60–62, 73, 95, 98, 103, 107, 112, 115, 116].

Regarding to DNN testing metrics, Pei et al. [73] proposed the metric of neuron coverage and a white-box testing framework based on this metric. Ma et al. [60] further proposed *Deepgauge*, a set of multi-granularity testing metrics for DNN. Kim et al. [45] proposed *SADL* to measure the test adequacy for DNN, including SA (Surprise Adequacy) and SC (Surprise Coverage). Du et al. [26] proposed two similarity metrics and five coverage criteria for stateful DL systems by modeling an RNN as Discrete-time Markov Chain. Gerasimou et al. [30] proposed the Importance-driven Coverage criterion to cover various combinations of the behaviors of important neurons. Sekhon and Fleming [87] proposed a coverage criterion to capture all possible parts of the logic of DNN. Also, Du et al. [27] and Ma et al. [59] proposed state coverage and t-way combination coverage for DNN, respectively.

Regarding to adversarial input generation, Xie et al. [103] proposed *Deephunter*, which is a coverage-guided fuzz testing framework for DNN. It generates new semantically retained test inputs through metamorphic testing. Guo et al. [35] proposed to maximize the neuron coverage and generate more adversarial inputs via mutation and differential testing. Sun et al. [94] utilized concolic testing to generate test inputs for DNN. Wang et al. [97] proposed to find adversarial inputs by proposing a measure of sensitivity and integrating statistical hypothesis and model mutation. Zhang et al. [113] proposed a condition-guided adversarial generative testing tool, called CAGTest, to efficiently generate test inputs for DNN, and CAGTest does not generate a large number of invalid test inputs. Currently, adversarial input generation has been widely used in various fields, including image recognition [9, 32, 38, 102], natural language processing [56, 72, 82], and speech recognition [74, 84, 106].

Different from them, our work aims to improve the efficiency of DNN testing by reducing labeling costs. More specifically, our work proposed PACE to select a small set of test inputs that can precisely estimate the accuracy of the whole testing set for DNN and then reduce labeling costs by just labeling this small set.

In addition, there are some related work to ours in DNN space. Active learning is a semi-supervised method, between unsupervised (i.e., 0% labeled data) and fully supervised (i.e., 100% labeled data) in terms of the amount of labeled data for ***training*** [67, 88]. More specifically, active learning aims to iteratively label a small set of data for ***training*** to *achieve similar (or greater) performance to using a fully supervised training set*. For example, Lewis and Gale [51] proposed to select the data for labeling, whose label has the largest uncertainty. Bouneffouf [8] proposed a sequential algorithm called exponentiated gradient (EG)-active to improve the selection of data for labeling by an optimal random exploration. However, different from active learning, our work aims to improve the DNN ***testing*** process by labeling a small number of test inputs to *precisely estimate the accuracy of the whole testing set*. That is, both goal and usage scenario between our work and active learning are totally different.

7.2 Test Optimization for Traditional Software

In traditional software testing [16, 58], test optimization is also an important direction to improve the efficiency of testing, including topics on test selection, test prioritization, and test-suite reduction. Please refer to a survey by Yoo and Harman for more details [104].

Test selection in traditional testing refers to selecting and running tests that are affected by software changes in regression testing, since the tests that are not affected by code changes should have the same results with previous runs [31, 50, 76–78, 109]. For example, Gligoric et al. [31] proposed a file-level dynamic test selection approach for Java projects based on the changes of bytecode class files, while Legunsen et al. [49] conducted an extensive study to compare different static test selection approaches. More recently, Zhang [108] proposed the first hybrid test selection approach by combining the strengths of existing dynamic test selection approaches at different granularities.

Test prioritization in traditional testing aims to prioritize all the tests to reveal software bugs as early as possible [7, 12–14, 28, 39, 58, 80, 81, 86, 105]. For example, Li et al. [53] proposed a search-based test prioritization approach by utilizing search-based algorithms to find the optimal order of test execution based on code coverage information. Jiang et al. [42] proposed an adaptive random test prioritization approach, which defines test distance to determine which test should be selected as the next one during prioritization. Chen et al. [20] conducted an empirical study to compare various test prioritization approaches and further proposed a machine-learning-based approach to recommending the optimal test prioritization approach for a specific project based on test distribution information.

Test-suite reduction aims to remove redundant tests from a test suite with respect to some testing requirements (e.g., code coverage) [15, 23, 24, 79, 90, 110]. For example, Harrold et al. [36] proposed a test-suite reduction approach by identifying the tests that are essential to cover some statements, called *essential tests*. Gotlieb and Marijan [34] proposed to remove tests from a test suite via searching among network maximum flows. Shi et al. [89] proposed to reduce a test suite by utilizing killed mutants.

Our work is more closely related to the topic of test-suite reduction, since our work also tries to reduce certain tests (although of different forms). Different from traditional test-suite reduction, our work aims to select a small set of test inputs that can represent the *accuracy* of the whole testing set for *deep neural network* testing, where the traditional test-suite reduction cannot apply due to the intrinsic differences between deep neural networks and traditional software systems.

8 CONCLUSION

To improve the efficiency of deep neural network (DNN) testing, we aim to select a small set of test inputs that can precisely estimate the accuracy of the whole testing set, and then just label

this small set instead of the whole testing set. To achieve this goal, we propose PACE, a novel and practical approach to selecting such a small set of test inputs. To make PACE more practical, PACE incorporates clustering to interpretably discriminate test inputs with different testing capabilities into different groups, and then utilizes the MMD-critic algorithm to interpretably select prototypes from each group and borrows the idea of adaptive random testing to select test inputs from the minority space (i.e., test inputs that are not clustered into any group) by considering diversity, to constitute the final small set of selected test inputs. We conducted an extensive study to evaluate the performance of PACE based on 24 pairs of DNN models and testing sets. This benchmark is comprehensive by considering different types of DNN models and different types of test inputs. The results demonstrate that PACE achieves great accuracy estimation effectiveness of only 1.181%~2.302% deviations on average, significantly outperforming all the compared approaches (i.e., SRS, CES, and CSS) with the average improvements of 51.06%, 50.94%, and 70.12%, respectively.

REFERENCES

[1] FastICA. 2019. Retrieved from https://scikit-learn.org/stable/modules/classes.html#module-sklearn.decomposition.

[2] HDBScan. 2019. Retrieved from https://pypi.org/project/hdbscan/.

[3] Keras. 2019. Retrieved from https://keras.io/.

[4] Github. 2019. MMD-critic algorithms. Retrieved from https://github.com/BeenKim/MMD-critic.

[5] scikit. 2019. scikit-learn. Retrieved from https://scikit-learn.org/stable/.

[6] Tensorflow. 2019. Retrieved from https://www.tensorflow.org/.

[7] Paul Ammann and Jeff Offutt. 2016. *Introduction to Software Testing*. Cambridge University Press.

[8] Djallel Bouneffouf. 2016. Exponentiated gradient exploration for active learning. *Computers* 5, 1 (2016), 1.

[9] Wieland Brendel, Jonas Rauber, and Matthias Bethge. 2018. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *Proceedings of the 6th International Conference on Learning Representations*.

[10] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'17)*. IEEE, 39–57.

[11] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. 2015. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*. IEEE, 2722–2730.

[12] Junjie Chen. 2018. Learning to accelerate compiler testing. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. 472–475.

[13] Junjie Chen, Yanwei Bai, Dan Hao, Yingfei Xiong, Hongyu Zhang, and Bing Xie. 2017. Learning to prioritize test programs for compiler testing. In *Proceedings of the 39th International Conference on Software Engineering*. 700–711.

[14] Junjie Chen, Yanwei Bai, Dan Hao, Yingfei Xiong, Hongyu Zhang, Lu Zhang, and Bing Xie. 2016. Test case prioritization for compilers: A text-vector-based approach. In *2016 IEEE International Conference on Software Testing, Verification and Validation*. 266–277.

[15] Junjie Chen, Yanwei Bai, Dan Hao, Lingming Zhang, Lu Zhang, and Bing Xie. 2017. How do assertions impact coverage-based test-suite reduction? In *Proceedings of the IEEE International Conference on Software Testing, Verification and Validation*. IEEE, 418–423.

[16] Junjie Chen, Yanwei Bai, Dan Hao, Lingming Zhang, Lu Zhang, Bing Xie, and Hong Mei. 2016. Supporting oracle construction via static analysis. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. 178–189.

[17] Junjie Chen, Xiaoting He, Qingwei Lin, Yong Xu, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2019. An empirical investigation of incident triage for online service systems. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*. 111–120.

[18] Junjie Chen, Xiaoting He, Qingwei Lin, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2019. Continuous incident triage for large-scale online service systems. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering*. 364–375.

[19] Junjie Chen, Wenxiang Hu, Lingming Zhang, Dan Hao, Sarfraz Khurshid, and Lu Zhang. 2018. Learning to accelerate symbolic execution via code transformation. In *Proceedings of the 32nd European Conference on Object-Oriented Programming*. 6:1–6:27.

[20] Junjie Chen, Yiling Lou, Lingming Zhang, Jianyi Zhou, Xiaoleng Wang, Dan Hao, and Lu Zhang. 2018. Optimizing test prioritization via test distribution analysis. In *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 656–667.

[21] Junjie Chen, Guancheng Wang, Dan Hao, Yingfei Xiong, Hongyu Zhang, Lu Zhang, and Xie Bing. 2018. Coverage prediction for accelerating compiler testing. *IEEE Trans. Softw. Eng.* (2018).

[22] Tsong Yueh Chen, Fei-Ching Kuo, Robert G. Merkel, and T. H. Tse. 2010. Adaptive random testing: The ART of test case diversity. *J. Syst. Softw.* 83, 1 (2010), 60–66.

[23] Tsong Yueh Chen and Man Fai Lau. 1998. A new heuristic for test suite reduction. *Info. Softw. Technol.* 40, 5–6 (1998), 347–354.

[24] Emilio Cruciani, Breno Miranda, Roberto Verdecchia, and Antonia Bertolino. 2019. Scalable approaches for test suite reduction. In *Proceedings of the 41st International Conference on Software Engineering*. IEEE Press, 419–429.

[25] Jianhua Dai and Qing Xu. 2013. Attribute selection based on information gain ratio in fuzzy rough set theory with application to tumor classification. *Appl. Softw. Comput.* 13, 1 (2013), 211–221.

[26] Xiaoning Du, Xiaofei Xie, Yi Li, Lei Ma, Yang Liu, and Jianjun Zhao. 2019. Deepstellar: Model-based quantitative analysis of stateful deep learning systems. In *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 477–487.

[27] Xiaoning Du, Xiaofei Xie, Yi Li, Lei Ma, Jianjun Zhao, and Yang Liu. 2018. Deepcruiser: Automated guided testing for stateful deep learning systems. *arXiv preprint arXiv:1812.05339* (2018).

[28] Sebastian Elbaum, Alexey G. Malishevsky, and Gregg Rothermel. 2002. Test case prioritization: A family of empirical studies. *IEEE Trans. Softw. Eng.* 28, 2 (2002), 159–182.

[29] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. 2016. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2414–2423.

[30] Simos Gerasimou, Hasan Ferit Eniser, Alper Sen, and Alper Cakan. 2020. Importance-driven deep learning system testing. *arXiv preprint arXiv:2002.03433* (2020).

[31] Milos Gligoric, Lamyaa Eloussi, and Darko Marinov. 2015. Practical regression test selection with dynamic file dependencies. In *Proceedings of the International Symposium on Software Testing and Analysis*. 211–222.

[32] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *Proceedings of the 3rd International Conference on Learning Representations*.

[33] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *Proceedings of the 3rd International Conference on Learning Representations*.

[34] Arnaud Gotlieb and Dusica Marijan. 2014. FLOWER: Optimal test suite reduction as a network maximum flow. In *Proceedings of the International Symposium on Software Testing and Analysis*. ACM, 171–180.

[35] Jianmin Guo, Yu Jiang, Yue Zhao, Quan Chen, and Jiaguang Sun. 2018. Dlfuzz: Differential fuzzing testing of deep learning systems. In *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 739–743.

[36] M. Jean Harrold, Rajiv Gupta, and Mary Lou Soffa. 1993. A methodology for controlling the size of a test suite. *ACM Transactions on Software Engineering and Methodology* 2, 3 (1993), 270–285.

[37] John A. Hartigan and Manchek A. Wong. 1979. Algorithm AS 136: A k-means clustering algorithm. *J. Roy. Stat. Soc. Series C (Appl. Stat.)* 28, 1 (1979), 100–108.

[38] Warren He, Bo Li, and Dawn Song. 2018. Decision boundary analysis of adversarial examples. In *Proceedings of the 6th International Conference on Learning Representations*.

[39] Christopher Henard, Mike Papadakis, Mark Harman, Yue Jia, and Yves Le Traon. 2016. Comparing white-box and black-box test prioritization. In *Proceedings of the IEEE/ACM 38th International Conference on Software Engineering*. IEEE, 523–534.

[40] Qiang Hu, Lei Ma, Xiaofei Xie, Bing Yu, Yang Liu, and Jianjun Zhao. 2019. DeepMutation++: A mutation testing framework for deep learning systems. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1158–1161.

[41] Md Johirul Islam, Giang Nguyen, Rangeet Pan, and Hridesh Rajan. 2019. A comprehensive study on deep learning bug characteristics. In *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 510–520.

[42] Bo Jiang, Zhenyu Zhang, W. K. Chan, and T. H. Tse. 2009. Adaptive random test case prioritization. In *Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering*. 233–244.

[43] Kyle D. Julian, Jessica Lopez, Jeffrey S. Brush, Michael P. Owen, and Mykel J. Kochenderfer. 2016. Policy compression for aircraft collision avoidance systems. In *Proceedings of the IEEE/AIAA 35th Digital Avionics Systems Conference*. IEEE, 1–10.

[44] Been Kim, Rajiv Khanna, and Oluwasanmi Koyejo. 2016. Examples are not enough, learn to criticize! Criticism for interpretability. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*. 2288–2296.

[45] 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*. IEEE Press, 1039–1049.
[46] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. 2017. Adversarial examples in the physical world. In *Proceedings of the 5th International Conference on Learning Representations Workshop Track*.
[47] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Recurrent convolutional neural networks for text classification. In *29th AAAI Conference on Artificial Intelligence*.
[48] Daniel D. Lee and H. Sebastian Seung. 2001. Algorithms for non-negative matrix factorization. In *Advances in Neural Information Processing Systems*. MIT Press, 556–562.
[49] Owolabi Legunsen, Farah Hariri, August Shi, Yafeng Lu, Lingming Zhang, and Darko Marinov. 2016. An extensive study of static regression test selection in modern software evolution. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 583–594.
[50] Owolabi Legunsen, August Shi, and Darko Marinov. 2017. STARTS: STAtic regression test selection. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. 949–954.
[51] David D. Lewis and William A. Gale. 1994. A sequential algorithm for training text classifiers. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. Springer, 3–12.
[52] Xia Li, Wei Li, Yuqun Zhang, and Lingming Zhang. 2019. Deepfl: Integrating multiple fault diagnosis dimensions for deep fault localization. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 169–180.
[53] Zheng Li, Mark Harman, and Robert M. Hierons. 2007. Search algorithms for regression test case prioritization. *IEEE Trans. Softw. Eng.* 33, 4 (2007), 225–237.
[54] Zenan Li, Xiaoxing Ma, Chang Xu, and Chun Cao. 2019. Structural coverage criteria for neural networks could be misleading. In *Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results*. IEEE Press, 89–92.
[55] Zenan Li, Xiaoxing Ma, Chang Xu, Chun Cao, Jingwei Xu, and Jian Lü. 2019. Boosting operational DNN testing efficiency through conditioning. In *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 499–509.
[56] Bin Liang, Hongcheng Li, Miaoqiang Su, Pan Bian, Xirong Li, and Wenchang Shi. 2018. Deep text classification can be fooled. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*. 4208–4215.
[57] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E. Alsaadi. 2017. A survey of deep neural network architectures and their applications. *Neurocomputing* 234 (2017), 11–26.
[58] Yiling Lou, Junjie Chen, Lingming Zhang, and Dan Hao. 2019. Chapter one—A survey on regression test-case prioritization. *Advances in Computers* 113 (2019), 1–46.
[59] 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 *Proceedings of the IEEE 26th International Conference on Software Analysis, Evolution and Reengineering*. IEEE, 614–618.
[60] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, et al. 2018. Deepgauge: Multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 120–131.
[61] Lei Ma, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Felix Juefei-Xu, Chao Xie, Li Li, Yang Liu, Jianjun Zhao, et al. 2018. Deepmutation: Mutation testing of deep learning systems. In *Proceedings of the IEEE 29th International Symposium on Software Reliability Engineering*. IEEE, 100–111.
[62] Shiqing Ma, Yingqi Liu, Wen-Chuan Lee, Xiangyu Zhang, and Ananth Grama. 2018. MODE: Automated neural network model debugging via state differential analysis and input selection. In *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 175–186.
[63] Shiqing Ma, Yingqi Liu, Guanhong Tao, Wen-Chuan Lee, and Xiangyu Zhang. 2019. NIC: Detecting adversarial samples with neural network invariant checking. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium*.
[64] Wei Ma, Mike Papadakis, Anestis Tsakmalis, Maxime Cordy, and Yves Le Traon. 2019. Test selection for deep learning systems. *CoRR* abs/1904.13195 (2019).
[65] Ke Mao, Mark Harman, and Yue Jia. 2016. Sapienz: Multi-objective automated testing for Android applications. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*. ACM, 94–105.
[66] Leland McInnes, John Healy, and Steve Astels. 2017. hdbscan: Hierarchical density-based clustering. *J. Open Source Softw.* 2, 11 (2017), 205.
[67] Prem Melville and Raymond J. Mooney. 2004. Diverse ensembles for active learning. In *Proceedings of the 21st International Conference on Machine Learning*. ACM, 74.

[68] Bernard M. E. Moret and Henry D. Shapiro. 1992. An empirical assessment of algorithms for constructing a minimum spanning tree. *Computational Support for Discrete Mathematics* 15 (1992), 99–117.

[69] Ziad Obermeyer and Ezekiel J. Emanuel. 2016. Predicting the future—Big data, machine learning, and clinical medicine. *New Engl. J. Med.* 375, 13 (2016), 1216.

[70] Augustus Odena, Catherine Olsson, David Andersen, and Ian J. Goodfellow. 2019. TensorFuzz: Debugging neural networks with coverage-guided fuzzing. In *Proceedings of the 36th International Conference on Machine Learning*. 4901–4911.

[71] Erkki Oja and Zhijian Yuan. 2006. The FastICA algorithm revisited: Convergence analysis. *IEEE Trans. Neural Netw.* 17, 6 (2006), 1370–1381.

[72] Nicolas Papernot, Patrick McDaniel, Ananthram Swami, and Richard Harang. 2016. Crafting adversarial input sequences for recurrent neural networks. In *Proceedings of the IEEE Military Communications Conference (MILCOM'16)*. IEEE, 49–54.

[73] 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*. ACM, 1–18.

[74] Yao Qin, Nicholas Carlini, Garrison W. Cottrell, Ian J. Goodfellow, and Colin Raffel. 2019. Imperceptible, robust, and targeted adversarial examples for automatic speech recognition. In *Proceedings of the 36th International Conference on Machine Learning*. 5231–5240.

[75] J. Ross Quinlan. 1986. Induction of decision trees. *Mach. Learn.* 1, 1 (1986), 81–106.

[76] Xiaoxia Ren, Fenil Shah, Frank Tip, Barbara G. Ryder, and Ophelia C. Chesley. 2004. Chianti: A tool for change impact analysis of Java programs. In *Proceedings of the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*. 432–448.

[77] Gregg Rothermel and Mary Jean Harrold. 1996. Analyzing regression test selection techniques. *IEEE Trans. Softw. Eng.* 22, 8 (1996), 529–551.

[78] Gregg Rothermel and Mary Jean Harrold. 1997. A safe, efficient regression test selection technique. *ACM Trans. Softw. Eng. Methodol.* 6, 2 (1997), 173–210.

[79] Gregg Rothermel, Mary Jean Harrold, Jeffery Von Ronne, and Christie Hong. 2002. Empirical studies of test-suite reduction. *Softw. Test. Verificat. Reliabil.* 12, 4 (2002), 219–249.

[80] Gregg Rothermel, Roland H. Untch, Chengyun Chu, and Mary Jean Harrold. 1999. Test case prioritization: An empirical study. In *Proceedings IEEE International Conference on Software Maintenance*. IEEE, 179–188.

[81] Ripon K. Saha, Lingming Zhang, Sarfraz Khurshid, and Dewayne E. Perry. 2015. An information retrieval approach for regression test prioritization based on program changes. In *Proceedings of the 37th IEEE/ACM International Conference on Software Engineering*. 268–279.

[82] Suranjana Samanta and Sameep Mehta. 2017. Towards crafting text adversarial samples. *arXiv preprint arXiv:1707.02812* (2017).

[83] Stefan Schaal, Sethu Vijayakumar, and Christopher G. Atkeson. 1998. Local dimensionality reduction. In *Advances in Neural Information Processing Systems*. MIT Press, 633–639.

[84] Lea Schönherr, Katharina Kohls, Steffen Zeiler, Thorsten Holz, and Dorothea Kolossa. 2019. Adversarial attacks against automatic speech recognition systems via psychoacoustic hiding. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium*.

[85] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. 2017. DBSCAN revisited, revisited: Why and how you should (still) use DBSCAN. *ACM Trans. Database Syst.* 42, 3 (2017), 19.

[86] Amanda Schwartz and Hyunsook Do. 2016. Cost-effective regression testing through adaptive test prioritization strategies. *J. Syst. Softw.* 115 (2016), 61–81.

[87] Jasmine Sekhon and Cody Fleming. 2019. Towards improved testing for deep learning. In *Proceedings of the IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER'19)*. IEEE, 85–88.

[88] Burr Settles and Mark Craven. 2008. An analysis of active learning strategies for sequence labeling tasks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 1070–1079.

[89] August Shi, Alex Gyori, Milos Gligoric, Andrey Zaytsev, and Darko Marinov. 2014. Balancing trade-offs in test-suite reduction. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 246–256.

[90] August Shi, Tifany Yung, Alex Gyori, and Darko Marinov. 2015. Comparing and combining test-suite reduction and regression test selection. In *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering*. ACM, 237–247.

[91] Qingkai Shi, Jun Wan, Yang Feng, Chunrong Fang, and Zhenyu Chen. 2019. DeepGini: Prioritizing massive tests to reduce labeling cost. *arXiv preprint arXiv:1903.00661*.

[92] Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the 3rd International Conference on Learning Representations*.

[93] Yi Sun, Yuheng Chen, Xiaogang Wang, and Xiaoou Tang. 2014. Deep learning face representation by joint identification-verification. In *Advances in Neural Information Processing Systems*. MIT Press, 1988–1996.

[94] 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*. ACM, 109–119.

[95] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *Proceedings of the 2nd International Conference on Learning Representations*.

[96] András Vargha and Harold D. Delaney. 2000. A critique and improvement of the CL common language effect size statistics of McGraw and Wong. *J. Edu. Behav. Stat.* 25, 2 (2000), 101–132.

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

[98] Zan Wang, Ming Yan, Shuang Liu, Junjie Chen, Dongdi Zhang, Zhuo Wu, and Xiang Chen. 2020. A survey on testing of deep neural networks. *J. Softw.* (2020). to appear.

[99] Pete Warden. 2017. Speech commands: A public dataset for single-word speech recognition. Retrieved from http://download.tensorflow. org/data/speech_commands_v0.

[100] Frank Wilcoxon. 1992. Individual comparisons by ranking methods. In *Breakthroughs in Statistics*. Springer, 196–202.

[101] Svante Wold, Kim Esbensen, and Paul Geladi. 1987. Principal component analysis. *Chemometr. Intell. Lab. Syst.* 2, 1–3 (1987), 37–52.

[102] Chaowei Xiao, Jun-Yan Zhu, Bo Li, Warren He, Mingyan Liu, and Dawn Song. 2018. Spatially transformed adversarial examples. In *Proceedings of the 6th International Conference on Learning Representations*.

[103] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. 2019. DeepHunter: A coverage-guided fuzz testing framework for deep neural networks. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, 146–157.

[104] Shin Yoo and Mark Harman. 2012. Regression testing minimization, selection and prioritization: A survey. *Softw. Test. Verification Reliability* 22, 2 (2012), 67–120.

[105] Zhe Yu, Jeffrey C. Carver, Gregg Rothermel, and Tim Menzies. 2019. Searching for better test case prioritization schemes: A case study of AI-assisted systematic literature review. *arXiv preprint arXiv:1909.07249*.

[106] Xuejing Yuan, Yuxuan Chen, Yue Zhao, Yunhui Long, Xiaokang Liu, Kai Chen, Shengzhi Zhang, Heqing Huang, XiaoFeng Wang, and Carl A. Gunter. 2018. Commandersong: A systematic approach for practical adversarial voice recognition. In *Proceedings of the 27th USENIX Security Symposium*. 49–64.

[107] Jie M. Zhang, Mark Harman, Lei Ma, and Yang Liu. 2020. Machine learning testing: Survey, landscapes and horizons. *IEEE Trans. Softw. Eng.* (2020).

[108] Lingming Zhang. 2018. Hybrid regression test selection. In *Proceedings of the IEEE/ACM 40th International Conference on Software Engineering*. IEEE, 199–209.

[109] Lingming Zhang, Miryung Kim, and Sarfraz Khurshid. 2011. Localizing failure-inducing program edits based on spectrum information. In *Proceedings of the 27th IEEE International Conference on Software Maintenance*. IEEE, 23–32.

[110] Lingming Zhang, Darko Marinov, Lu Zhang, and Sarfraz Khurshid. 2011. An empirical study of junit test-suite reduction. In *Proceedings of the IEEE 22nd International Symposium on Software Reliability Engineering*. IEEE, 170–179.

[111] Long Zhang, Xuechao Sun, Yong Li, and Zhenyu Zhang. 2019. A noise-sensitivity-analysis-based test prioritization technique for deep neural networks. *arXiv preprint arXiv:1901.00054*.

[112] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. 2018. Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 132–142.

[113] Pengcheng Zhang, Qiyin Dai, and Shunhui Ji. 2019. Condition-guided adversarial generative testing for deep learning systems. In *Proceedings of the IEEE International Conference On Artificial Intelligence Testing (AITest'19)*. IEEE, 71–72.

[114] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, Junjie Chen, Xiaoting He, Randolph Yao, Jian-Guang Lou, Murali Chintalapati, Furao Shen, and Dongmei Zhang. 2019. Robust log-based anomaly detection on unstable log data. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 807–817.

[115] Zhiyi Zhang and Xiaoyuan Xie. 2019. On the investigation of essential diversities for deep learning testing criteria. In *Proceedings of the IEEE 19th International Conference on Software Quality, Reliability and Security (QRS'19)*. IEEE, 394–405.

[116] Husheng Zhou, Wei Li, Yuankun Zhu, Yuqun Zhang, Bei Yu, Lingming Zhang, and Cong Liu. 2020. Deepbillboard: Systematic physical-world testing of autonomous driving systems. In *Proceedings of the International Conference on Software Engineering (ICSE'20)*.

Received October 2019; revised April 2020; accepted April 2020