Cost-Effective Testing of a Deep Learning Model through Input Reduction

Jianyi Zhou*†, Feng Li*†, Jinhao Dong*†, Hongyu Zhang‡, Dan Hao*†§
*Key Laboratory of High-Confidence Technologies (Peking University), MoE
†Institute of Software, Department of Computer Science, EECS, Peking University
{zhoujianyi,lifeng2014}@pku.edu.cn, jhdong@stu.pku.edu.cn, haodan@pku.edu.cn
‡School of Electrical Engineering and Computing
The University of Newcastle, Callaghan, NSW 2308, Australia
hongyu.zhang@newcastle.edu.au

***Abstract*—With the increasing adoption of Deep Learning (DL) models in various applications, testing DL models is vitally important. However, testing DL models is costly and expensive, e.g., manual labelling is widely-recognized to be costly. To reduce testing cost, we propose to select only a subset of testing data, which is small but representative enough for a quick estimation of the performance of DL models. Our approach, DeepReduce, adopts a two-phase strategy. At first, our approach selects testing data for the purpose of satisfying testing adequacy. Then, it selects more testing data to approximate the distribution between the whole testing data and the selected data by leveraging relative entropy minimization. We evaluate DeepReduce on four widely-used datasets (with 15 models in total). We find that DeepReduce reduces the whole testing data to 7.5% on average and can reliably estimate the performance of DL models.**

***Index Terms*—Software Testing, Deep Learning, Input Data Reduction**

I. Introduction

Nowadays, deep learning (DL) models have been widely deployed in various applications. However, like traditional software, DL-based applications contain faults too [1], [2]. In recent years, software testing has been applied to ensure the reliability of DL models [3]–[5].

However, DL model testing is very costly due to the extremely large number of testing data. For example, DeepFace [6], the face recognition system of Facebook, used about 0.22 million face images for testing. DeepTest [7] generated 254,221 images (whose neuron coverage, the ratio of activated neurons, is 88%) for testing a Chauffeur-CNN based autonomous driving model. The popular ImageNet dataset [8] contains 100,000 testing images (100 per class) for testing various image classification models. Such a large number of testing data with oracle information may guarantee that the DL model is sufficiently tested, but will also consume much testing time. In particular, manually labeling those data is very time-consuming. Moreover, the testing data may be repeatedly executed for evaluating alternative designs of DL models and optimizing the models, which can be regarded as a typical regression scenario and leads to an increased cost in testing.

To reduce the testing cost, it is desirable to have an early estimation of the performance of a DL model by using less testing data. A small amount of testing data mean less manual labeling cost. Besides, in regression testing, executing a reduced amount of testing data also results in less cost than executing the entire testing data. If this small subset reveals that the model performance is acceptable, developers can then perform a full-scale testing. Otherwise, developers can continue to seek alternative designs of the model (e.g., adding/deleting a layer) or further tune the hyper-parameters using the same subset. That is, with the small amount of testing data, developers can quickly obtain an accurate DL model without having to perform costly full-scale testing many times.

To achieve an early estimation of the performance of a DL model, this paper defines an input reduction problem, which is to *select a small yet effective subset of testing data* for cost-effective testing of DL models. The set of selected data is expected to achieve similar testing performance (which is characterized by testing completeness and testing effectiveness) as the entire testing data set, and to be small enough to reduce the testing cost. To our knowledge, this is the first piece of work on testing data reduction with the purpose of early performance estimation. Recently, Li et al. [9] proposed an input selection approach CES to reduce the number of testing data required to be labelled so as to reduce the manual efforts in DL testing, which is very related to our work. Both this work and CES share the same general goal, alleviating the cost problem of DL testing, but CES does not require the selected testing data to be complete. More technical difference and performance comparison between this work and CES are given later in Sections III-C and IV.

In particular, we formulate the input reduction problem in this work as a multi-objective optimization problem for reliable performance estimation: 1) *minimize* the amount of testing data selected from the whole testing data (efficiency); 2) *maximize* the testing adequacy achieved by the selected testing data (completeness); and 3) *maximize* the similarity of output distributions achieved by the selected and the whole testing data (effectiveness). Testing efficiency, demonstrated by the testing cost in this work, is measured by the amount of testing data required in the testing process. Note that testing effectiveness is usually measured by testing accuracy, which is the ratio of correctly predicted testing data. However,

§Corresponding author.

testing accuracy is not always available due to the difficulty in testing data labelling. Therefore, in this work, we use the similarity of output distribution (which can be viewed as a small probability metric/distance [10]) to measure testing effectiveness, because similar output distribution usually leads to close testing accuracy.

The challenge in this problem lies in how to balance these relevant (even contradictory) objectives, which has never been studied by existing work on DL testing. In particular, we present a two-phase reduction approach DeepReduce, which satisfies both the testing adequacy and the similarity of output distributions. To measure the testing adequacy, we use neuron coverage criteria in this work, which is a widely used criteria for measuring testing adequacy in DL testing. To represent output distribution, we use the outputs of the neurons in the last layer of a DL model, which is more related to the behavior of a DL model [11], [12]. Guided by these two metrics, DeepReduce first selects a minimized subset of testing data with sufficient testing adequacy through a greedy strategy, and adds more testing data to the subset based on relative-entropy minimization through a heuristic-based method.

To evaluate our approach DeepReduce, we conducted extensive experiments on 15 DL models of four datasets. The experiment results show that the absolute difference between the accuracy achieved by the testing data selected by DeepReduce and the original is 0.0062 on average, indicating that DeepReduce performs well in estimating the performance of DL models. Besides, DeepReduce reduces the amount of testing data required for a reliable testing to 7.5% on average, which means that over 92.5% testing cost can be saved by DeepReduce. Compared with CES [9], which is the state-of-the-art technique for reducing manual efforts in DL testing, when selecting the same number of testing data, CES achieves 0.0215 in terms of accuracy difference on average. That is, DeepReduce outperforms CES in terms of performance estimation. DeepReduce is also more stable and reliable, and can reduce more testing data according to our evaluation results. Besides, we also conducted a preliminary study on the performance of DeepReduce in regression testing, i.e., the estimation performance of selected testing data on modified models, and found that DeepReduce achieves an average of 0.0104 in terms of accuracy difference, indicating that DeepReduce can be also used to mitigate regression testing cost to some degree.

In summary, the contributions of this paper are as follows:

- A novel approach to cost-effective testing a DL model by considering three objectives: efficiency, completeness, and effectiveness.
- An empirical evaluation to investigate the performance of our approach on real-world DL models. The results have confirmed that the proposed approach can significantly reduce DL testing cost by reducing the amount of testing data required for testing a DL model.

II. Proposed Approach

In order to estimate the performance of a DL model in a cost-effective way, this work attempts to select a representative subset of testing data from a given set of testing data.

A. Problem Description

Considering the usage, the selected data are expected to satisfy at least two objectives. The first one is that the selected data are expected to have the same testing adequacy (e.g., neuron coverage) as the given set, so as to guarantee that the performance estimation is conducted on the complete learning model, instead of a partial learning model. The second one is that the selected data are expected to generate the same output distribution as the given set, so as to guarantee that the estimation is correct and precise. That is, this paper targets an input reduction problem, which is formally defined as:

Definition 1. *Given a set of testing data T and its target DL model M, we define function $testReq(T, M)$ to measure to what extent the testing data T satisfy the testing requirement of M and function $outDist(T, M)$ to measure the output distribution of M with respect to T. The problem of input reduction in DL testing is to find the smallest subset of testing data from T, denoted as T', satisfying that $testReq(T, M) = testReq(T', M)$ and $outDist(T, M) = outDist(T', M)$.*

As the input reduction problem is NP-complete, we relax its requirements on the output distribution $outDist(T', M)$ to make it easier to find possible solutions. That is, we replace $outDist(T, M) = outDist(T', M)$ by $outDist(T, M) \simeq outDist(T', M)$, i.e., the output distribution $outDist(T, M)$ is similar to $outDist(T', M)$. Such a requirement on output distribution guarantees that the estimation using T' is close to or even the same as T on M. Note that through this relaxing process, the input reduction problem is still NP-complete. However, with this relaxing process, more solutions instead of one (i.e., a subset of testing data whose output distribution is close to the original set) exist so that the redefined input reduction problem seems to be easier, especially considering the measurement used in the paper.

To solve the redefined input reduction problem, i.e., finding a smaller set instead of the smallest set, we present a new technique DeepReduce to select a small set of testing data for cost-effective testing. That is, given a testing data set T and its learning model M, DeepReduce iteratively selects a candidate testing data until the two objectives are satisfied. In particular, we first present the metrics used in DeepReduce to define functions $testReq$ and $outDist$ (in Section II-B), and then a two-phase reduction algorithm used in DeepReduce to select testing data (in Section II-C).

B. Testing Adequacy and Output Distribution

Testing Adequacy. Structural coverage is widely used for measuring testing adequacy in conventional software testing, which measures to what extent structural elements (e.g., statements and methods) are covered by a test or test suite. It is usually taken as the default requirement for software testing. In the literature, there are various neuron coverage criteria (e.g., Neuron Coverage (abbreviated as NC) [13], k-Multisection Neuron Coverage and Neuron Boundary Coverage [14]) proposed in DL testing, each of which can be used to define the function $testReq$. Here, we use NC as the example to illustrate our input reduction approach, because it is lightweight and widely used/studied in the literature [3], [5], [13]. NC [13] is defined as the ratio of activated neurons of a DL model, where an activated neuron refers to the neuron whose output of an input data is larger than a given threshold β. NC with various threshold β is taken as different coverage criteria.

Output Distribution. We measure the data distribution by leveraging the outputs of the neurons in the last layer since they are more related to the behavior of a DL model and the problem domain [11], [12]. For each neuron we record its outputs of all testing data, and divide its output range into several sections, each of which contains the same number of unique outputs. We then measure the data distribution across these sections. In particular, suppose that the target DL model M consists of m neurons in its last layer, we first record the outputs of the testing data set T on each neuron n_i ($i \in [1, m]$). For each neuron, we sort the unique values of its outputs, and group these *unique* values (following the order) into K sections (denoted as $D_{n_i,1}$, $D_{n_i,2}$, ..., and $D_{n_i,K}$) of equal size. In particular, $D_{n_i,k}(k \in [1, K-1])$ contains the unique values with the index of $\lfloor\frac{|T|*(k-1)}{K}\rfloor + 1$ to $\lfloor\frac{|T|*k}{K}\rfloor$, while $D_{n_i,K}$ contains the remaining values. The output of each data t ($t \in T$) on n_i falls into one of these sections. For each n_i, we calculate the ratio of testing data whose corresponding output falls into $D_{n_i,1}$, $D_{n_i,2}$, ..., and $D_{n_i,K}$, and denote the corresponding results by $P_T^{n_i} = \{P_T^{n_i}(1), ..., P_T^{n_i}(K)\}$, where $P_T^{n_i}(j)$ is the percentage of data in T whose corresponding output falls into $D_{n_i,j}$. Finally, the output distribution of M with respect to T is represented as $P_T : \{P_T^{n_1}, P_T^{n_2}, ..., P_T^{n_m}\}$.

For example, suppose the outputs of T on one neuron n_i in the last layer of M are {1,2,3,9,4,7,8,1,10} and K is 2, we first divide the sorted unique values into two equal-size sections, i.e., {1,2,3,4} and {7,8,9,10}, and then calculate the percentage of data falling into each section, which is 55.6% (5/9) and 44.4% (4/9) respectively. Therefore, $P_T^{n_i} = \{55.6\%, 44.4\%\}$.

C. Two-Phase Reduction Algorithm

With the $testReq$ and $outDist$ introduced in Section II-B, DeepReduce aims to select testing data by satisfying the two mentioned requirements. Algorithm 1 presents the two-phase reduction algorithm, where the input consists of testing set denoted as T, a DL model denoted as $\mathcal{M}$, coverage denoted as Cov, output distribution of T denoted as *TD*, output section of each data in T denoted as *TI*, the number of neurons denoted as m, the number of sections in each neuron denoted as K, and termination criterion denoted as α, respectively. The algorithm first selects a subset of T to guarantee the neuron coverage by using the HGS algorithm [15], and then selects more testing data with the purpose of maximizing the similarity of output distribution between T and T'.

In the first phase, this algorithm reuses the HGS [15] algorithm to select the minimized subset of T with the same NC coverage. In particular, the HGS algorithm is a greedy algorithm which tends to select testing data covering the activated neurons that are less covered by the existing testing data. In Line 2, *HGS*(T, Cov) is used to represent the output of the HGS algorithm on T by satisfying the NC coverage recorded by Cov, which is a subset of selected testing data. In Line 3, a function Update(*TD*$'$, T') is used to calculate the output distribution of the selected data set T' (denoted as *TD*$'$) by using the section division generated by *TD*, to facilitate the output comparison between T and T'. In Line 4, the remaining unselected testing data are put into *REMAIN*, which are to be selected in the second phase.

In the second phase, this algorithm selects more testing data (in Lines 6-40) until the set of selected data is similar to T in output distribution by using a heuristic-based method. That is, more testing data are selected until the termination

Algorithm 1: Two-Phase Reduction Algorithm

```
Input  : $T$, $\mathcal{M}$, $Cov$, $TD$, $TI$, $m$, $K$, $\alpha$
Output: Reduced testing data $T'$
1  // First Phase;
2  $T' \leftarrow HGS(T, Cov)$;
3  $TD' \leftarrow$ Update($TD'$, $T'$);
4  $REMAIN \leftarrow T \setminus T'$;
5  // Second Phase;
6  $KLlist \leftarrow \emptyset$ ;
7  while True do
8      $re \leftarrow \emptyset$;
9      foreach $i \in \{1, m\}$ do
10         $kmax \leftarrow 1$;
11         $maxvalue \leftarrow TD_{i,1}/TD'_{i,1}$;
12         foreach $k \in \{2, K\}$ do
13             if $maxvalue < TD_{i,k}/TD'_{i,k}$ then
14                 $kmax \leftarrow k$;
15                 $maxvalue \leftarrow TD_{i,k}/TD'_{i,k}$;
16         end
17         $re \leftarrow re \cup \{kmax\}$;
18     end
19     $maxsim \leftarrow 0$, $dict \leftarrow \emptyset$;
20     foreach $t \in REMAIN$ do
21         $sim \leftarrow$ getSimilarity($TI[t]$, $re$);
22         if $sim > maxsim$ then
23             $maxsim \leftarrow sim$;
24             $dict \leftarrow \{t\}$;
25         else if $sim == maxsim$ then
26             $dict \leftarrow dict \cup \{t\}$;
27     end
28     $minvalue \leftarrow \infty$;
29     foreach $t \in dict$ do
30         if $minvalue > KL(T, T' \cup \{t\})$ then
31             $minvalue \leftarrow KL(T, T' \cup \{t\})$;
32             $t' \leftarrow t$;
33     end
34     $T' \leftarrow T' \cup \{t'\}$;
35     $REMAIN \leftarrow REMAIN \setminus \{t'\}$;
36     $TD' \leftarrow$ Update($TD'$, $T'$);
37     $KLlist \leftarrow KLlist \cup \{KL(TD, TD')\}$;
38     if $KL(TD, TD') < \alpha$ then
39         break;
40 end
41 return $T'$;
```

criterion (i.e., the relative entropy value between T and T' is smaller than the specified threshold α) is satisfied, as shown by Lines 38-39. In particular, Lines 8-18 are to find the section in which the output distribution differs the most between T' and T on each neuron, and these sections are recorded based on the order of the corresponding neuron by re. For each neuron e_i, the algorithm calculates the output difference on each section $D_{e_i,k}$ between T and T' through $\mathit{TD}_{i,k}/\mathit{TD}'_{i,k}$, so as to find the section with the largest output difference on each neuron (denoted as $kmax$ in the algorithm). Then Lines 19-33 are to select the testing data t' that may minimize the output difference between T and the selected testing data set, where variable re is used to represent the output distribution difference between T and T', and the function *getSimilarity* is used to calculate the similarity between the output distributions. The variable *REMAIN* is a set recording unselected testing data. As shown by Lines 19-27, for each t in *REMAIN*, this algorithm iteratively calculates the number of neurons in *TI*$[t]$ and re with the same section, and finds the testing data with the largest number of such neurons. Note that $dict$ is an array recording candidate testing data, each of which has the largest similarity with re. Among all the candidates in $dict$, in Lines 28-33, this algorithm determines which testing data should be chosen by iteratively calculating the relative entropy between the given testing data set T and the selected testing data set by including each candidate. Lines 9-33 are to find the testing data contributing the most to the similarity of output distribution. Based on the selected testing data t, Lines 34-36 are to add the selected testing data into T', remove it from *REMAIN*, and update the data distribution *TD*$'$. Finally, Lines 37-39 are to calculate the relative entropy value between T and T', and check whether this value is smaller than the specified α, which is the termination criterion of this algorithm. More details about the calculation of the relative entropy value is given below.

For a learning model M with the given testing data set T, we use P_T to represent its output distribution, which is calculated by the ratio of testing data falling into sections $D_{e_i,j} (i \in [1,m], j \in [1,K])$. For the selected testing data set T', we calculate its output distribution on the sections $D_{e_i,j} (i \in [1,m], j \in [1,K])$ produced by T instead of T' for ease of comparison, which is denoted by $P_{T'}$. We use relative entropy [16], which is also called Kullback-Leibler Divergence, to measure the similarity between them (denoted as $\mathit{KL}(P_T, P_{T'})$). The calculation is defined in Formula 1. Relative entropy is often used to measure how one output distribution is different from the other one. Note that the output distribution of T or T' contains m probability distributions, we take their average as the similarity between two output distributions. The smaller the KL value is, the more similar these two output distributions are.

$$\mathit{KL}(P_T, P_{T'}) = \frac{\sum_{i=1}^{m} \sum_{j=1}^{K} P_T^{e_i}(j) * \log \frac{P_T^{e_i}(j)}{P_{T'}^{e_i}(j)}}{m} \qquad (1)$$

The algorithm complexity is $O(m * K * |T|) + O(m * |T|^2)$, where K denotes the number of sections in each neuron, m

TABLE I: The information for datasets.

Dataset	Model	Model Info			
		#Layers	#Neurons	#Tests	Acc
CIFAR-10	NIN	24	3,432	10,000	0.8815
	VGG19	65	50,782		0.9346
	ResNet	113	4,138		0.9220
	WideResNet	94	33,178		0.9537
	DenseNet	350	37,168		0.9463
MNIST	Lenet1	7	52	10,000	0.9754
	Lenet4	8	148		0.9848
	Lenet5	9	268		0.9858
CIFAR-100	NIN	25	3,592	10,000	0.5698
	VGG19	57	18,696		0.6159
	ResNet50	175	93,192		0.5541
	GoogleNet	127	27,464		0.6939
IMAGENET	ResNet50	178	95,059	50,000	0.7470
	VGG16	24	15,888		0.7127
	VGG19	27	17,168		0.7126

denotes the number of neurons in the last layer and $|T|$ denotes the number of testing data. Note this complexity is achieved only when $T = T'$, indicating no input reduction occurs. Besides, K is usually set to 20 in the literature [9] and m is usually not large in practice. Therefore, the complexity of DeepReduce is acceptable.

III. Experiment Setup

A. Datasets and DL Models

In this work, we used four widely-used image recognition datasets, MNIST [17], CIFAR-10, CIFAR-100 [18], and IMAGENET [19]. MNIST contains 60,000 training images and 10,000 testing images in total. We trained three LeNet family models (LeNet1, LeNet4, and LeNet5) [17] and evaluated DeepReduce on them. CIFAR-10 contains 50,000 training images and 10,000 testing images. On this dataset we trained five DL models, including NIN [20], VGG19 [21], ResNet [22], WideResNet [23], and DenseNet [24]. CIFAR-100 is just like the CIFAR-10, except it has 100 classes each of which contains 600 images (500 training images and 100 testing images). We trained four DL models, including NIN, VGG19, ResNet50, and GoogleNet [25]. IMAGENET collects more than 1.4 million images as training data, 50,000 images as validation data, and 50,000 images as testing data. It divides all images into 1,000 classes. We trained three DL models, i.e., ResNet, VGG16 and VGG19 on this dataset. To sum up, we collect 15 DL models on four data-sets in total. Table I presents the information of these models, where Columns 3-6 presents the basic statistics, and along with their testing accuracy obtained by the whole testing data. In particular, each of the first three datasets contains 10,000 testing data respectively, while IMAGENET contains 50,000 testing data.

B. Implementations

The implementations of all the studied DL models are collected from GitHub. More specifically, we used the default parameters in the script in training process. As recording the neuron coverage of all the testing data on IMAGENET takes up too much memory, we conducted the HGS (the first phase) in batch when implementing DeepReduce. Although such operation does not yield optimal results in test reduction, the testing adequacy can still be satisfied. All the scripts used in this experiment are written in Python. The empirical study was conducted on a workstation with 32-core Intel Xeon CPU E5-2683-v4(2.10GHz), 128G memory, and also two NVIDIA GPU (TITAN RTX (24G) and TITAN XP (12G)). For reproducibility and future usage, we put all the materials of this experiment on our project website: **https://github.com/DeepReduce/DeepReduce**.

C. Baseline Approaches

The most related work on input reduction is CES [9][1], which is used to select representative testing data from unlabelled data. However, it requires the number of testing data to be selected as the input of the algorithm, and it relies on the data distribution of the last hidden layer instead of the last layer. To enable comparison, we took the number of testing data selected by DeepReduce as an input of CES. Besides, we also implemented a variant of CES (denoted as CES$_{kl}$), by changing the termination criterion to be the KL value to align with our approach. That is, CES$_{kl}$ iteratively selects testing data until the KL value is smaller than α. As the redundancy of testing data in different datasets is different, we do not know how many testing instances are representative enough for a given DL model along with its dataset beforehand. Therefore, the KL value is more suitable than the number of testing data to be selected as the termination criterion.

Besides, we also implemented two random reduction approaches, i.e., RANDOM and RANDOM$_{kl}$. RANDOM randomly selects testing data from the whole testing data until the number of the selected data is larger than a given value (i.e., the number of testing data selected by DeepReduce), while RANDOM$_{kl}$ randomly selects testing data from the whole testing data until the KL value is smaller than α.

In summary, we implemented four baseline approaches. To reduce the influence of their inherent randomness, CES and CES$_{kl}$ are repeated 10 times, and the two random approaches are repeated 50 times. Their average results are used for comparison.

D. Neuron Coverage Criteria

In this experiment, we investigated the performance of DeepReduce by using various neuron coverage criteria proposed in the literature. In particular, the neuron coverage criteria studied in this experiment include the following.

Neuron Coverage (NC) [13]: Given a testing data, a neuron is activated if its output is greater than a given threshold, otherwise the neuron is non-activated. NC is defined by the ratio of activated neurons of a DNN, where the activation threshold is set to be 0.25, 0.5, or 0.75 in this paper. NC with various threshold is taken as different coverage criteria.

Neuron Boundary Coverage (NBC) [14]: For each neuron of a DNN, the outputs of all the training data on this neuron are located in an interval denoted as [*low*, *high*]. Given a testing data, if its output is not in this interval (i.e., $(-\infty$, *low*] and [*high*, $+\infty$)), it is regarded to cover the corner-case region of this neuron. NBC measures to what extent the corner-case region is covered.

[1]Li et al. [9] proposed two approaches CSS and CES. We took CES as the baseline because it is more effective according to their evaluation results.

Strong Neuron Activation Boundary Coverage (SNAC) [14]: SNAC is similar to NBC, but it considers only the upper corner-case region (i.e., [*high*, $+\infty$)) covered by testing data.

Top-k Neuron Coverage (TKNC) [14]: TKNC is defined as the ratio between the total number of top-k neurons in each layer and the total number of neurons in a DNN, where top-k neurons refers to the neurons used to be the most active k neurons in each layer. Given a testing data, it is regarded to cover the top-k neuron if and only if its output of the neuron is no less than the k_{th} highest value of all the neurons in the corresponding layer.

E. Research Questions and Experimental Design

We investigate the performance of the proposed approach from the following aspects.

RQ1. Performance: How does DeepReduce perform in input reduction?

To answer this RQ, we compared the performance of the proposed approach with the four baseline approaches, and measured (1) the reduction performance through the amount of the selected testing data, (2) the reduction ratio, and (3) the performance of selected testing data through the absolute difference between the testing accuracy achieved by the selected testing data and the original one. In particular, the testing accuracy is the ratio of correctly predicted testing data. More specifically, in this RQ, we used the NC with threshold $\beta = 0.5$ (abbreviated as NC(0.5)) and the termination criterion with threshold $\alpha = 0.001$ as the default setting.

RQ2. Parameter Influence: How do different parameters (i.e., coverage criteria and termination criteria) influence the performance of DeepReduce?

In this RQ, we studied 8 neuron coverage criteria in total, which are NC (setting β to 0.25, 0.5, and 0.75), NBC, SNAC, and TKNC (setting k to 1, 2, and 3). More specifically, we studied the influence caused by various neuron coverage criteria on three datasets excluding IMAGENET, as it is quite time-consuming to collect some coverage criteria due to the large training set. Besides, we studied the influences caused by various termination criteria, e.g., the threshold α on the KL value is set to be 0.05, 0.01, 0.005, and 0.001.

RQ3. Sensitivity Analysis: How do the components of DeepReduce affect its performance?

For sensitivity analysis, we investigated the impacts of components in the two-phase algorithm of DeepReduce . In particular, besides the two selection heuristics used in the two phases, which layer is used to get the output of the DL model and how to divide the output into sections (i.e., division strategy) may also influence the results of DeepReduce as they are components of the second phase. Therefore, we implemented the following variants of DeepReduce by considering the influence of these components.

1) We excluded the first phase from DeepReduce (abbreviated as Variant$_1$);
2) We used the output of the neurons in the last-hidden layer and the last layer to investigate the influence of different layers on DeepReduce (abbreviated as Variant$_2$);

3) We replaced the division strategy for output distribution with equally partitioning the output range (abbreviated as Variant$_3$).
4) We replaced the selection heuristic in the second phase with random selection (abbreviated as Variant$_4$).

More specifically, we used a fixed number of selected testing data as the termination criterion of each variant. That is, each variant iteratively selects testing data until the selected data is larger than a given value (i.e., the number of testing data selected by DeepReduce when setting α to 0.001 and the coverage criteria to NC(0.5)). Due to space limit, we show only sensitivity results on CIFAR-10. More sensitivity results on other datasets are presented on our website.

RQ4. Application in Regression Testing: How does DeepReduce perform in regression testing?

To mitigate the testing cost, our work can be used to get an early estimation of the performance of modified models, which is also a use case of our work. In this RQ, we presented a preliminary study of DeepReduce in the regression testing scenario. To simulate regression testing, we first collected a set of modified models, which are obtained through modification on the models and retraining process. In particular, we applied seven types of modification on the studied models, i.e., adding layer (ADL), deleting layer (DEL), adding neurons (ADN), deleting neurons (DEN), changing learning rate (LR), changing momentum (MO), and changing dropout ratio (DR). Although these changes may not be representative in practice, they are some typical changes used by the developers in practice, and have important influences on the performance of DNN models. Based on the modified models, we evaluated our approach by comparing the accuracy of the selected data against the whole testing data, hoping that the selected data set are still able to estimate the performance of modified models. More specifically, we reused the testing data selected by DeepReduce in RQ1, and calculated the accuracy and ΔAcc on all the modified models.

F. Threats to Validity

The internal threats to validity lie in the implementations of DeepReduce, the baseline approaches, and evaluation scripts. To reduce these threats, the first two authors reviewed the implementation code and scripts used in this work.

The external threats to validity mainly lie in the DL models, the datasets, and the used neuron coverage criteria. To reduce the first two threats, we used 4 datasets and various DL models, which are widely used in image recognition. The last threat comes from the coverage criteria used in this paper, since our approach is not specific to the eight neuron coverage criteria used in this work. To address these threats, in the future, we will use more datasets (e.g., autonomous driving dataset) and more criteria to evaluate the performance of DeepReduce.

The construction threat to validity mainly lies in the model modifications applied in regression testing. To reduce this threat, we designed seven changes in total, including structural and non-structural changes. All of these are typical changes used in model designing and hyper-parameters tuning. However, these changes are not representative of all the alternative designs of a model, e.g., complex and multiple changes are missing. In the future, we will evaluate DeepReduce by using more modifications in regression testing.

IV. Results Analysis

A. Results for RQ1

In this section, we analyze the performance of DeepReduce with $\alpha = 0.001$ against the baseline when the coverage criterion is NC(0.5).

1) Comparison using the same number of selected testing data: In Table II, we present the results of DeepReduce, CES, and RANDOM, each of which is required to select the same number of testing data. The column "Acc" represents the test accuracy achieved by the selected testing data, "ΔAcc" represents the absolute difference between the accuracy achieved by the selected testing data and the original one, "Size" represents the amount of selected testing data, "Ratio" represents the reduction ratio. "KL" represents the KL value between the selected testing data and the whole testing data, which is calculated based on Formula 1. CES and RANDOM take the results in "Size" (produced by DeepReduce) as input.

From Table II, DeepReduce reduces the whole testing data significantly, leaving only 2.5% to 17.9% testing data, with an average of 7.5%. Meanwhile, it achieves an average of 0.0063 in terms of ΔAcc, indicating that the effectiveness of DeepReduce in performance estimation is good. Even in the worst case, the ΔAcc is up to 0.0181. Compared with CES, DeepReduce outperforms it in terms of ΔAcc on 13 (out of 15) models. We emphasize the winner of the comparison between DeepReduce and CES through the bold font in the table.

Besides, the average ΔAcc of CES is 0.0215, while the average ΔAcc of DeepReduce is only 0.0063, indicating that the improvement of DeepReduce over CES is about 71.2% on average. We also conducted *Wilcoxon Signed-Rank Test* [26] (with 0.05 significance level) on the results of ΔAcc. The *p-value* is 0.0009, indicating that DeepReduce significantly outperforms CES in terms of ΔAcc. Since CES has its own reduction metric "KL" (i.e., different division strategies), we also present the results of its KL value in Column "KL". From the table, the KL values achieved by CES are still much larger than the ones achieved by DeepReduce (0.001), indicating that the similarity of output distribution achieved by CES is worse than the one achieved by DeepReduce.

Regarding to the comparison between DeepReduce and RANDOM, DeepReduce also outperforms RANDOM in terms of both ΔAcc and KL values. The *Wilcoxon Singed-Rank Test* also shows that the results is significant (whose *p-value* is 0.0332), i.e., DeepReduce significantly outperforms RANDOM in terms of ΔAcc. Considering that the results of RANDOM may vary a lot in practice, e.g., the ΔAcc of RANDOM varies from 0.0009 to 0.0511 on NIN (CIFARA-10), DeepReduce is more practical and effective than RANDOM.

DeepReduce also outperforms CES in terms of generality. Note that the number of reduced testing data needed for an

TABLE II: Model information and the effectiveness of DeepReduce (using the same number of selected testing data).

Dataset	Model	Original Acc	DeepReduce				CES			RANDOM		
			Acc	Δacc	Size	Ratio	Acc	Δacc	KL	Acc	Δacc	KL
CIFAR-10	NIN	0.8815	0.8754	**0.0061**	289	2.9%	0.8948	0.0133	0.0606	0.8796	0.0145	0.0326
	VGG19	0.9346	0.9249	**0.0097**	493	4.9%	0.9118	0.0228	0.0287	0.9328	0.0094	0.0190
	ResNet	0.9220	0.9194	**0.0026**	273	2.7%	0.9414	0.0194	0.0533	0.9253	0.0137	0.0361
	WideResNet	0.9537	0.9491	**0.0046**	373	3.7%	0.9456	0.0081	0.0532	0.9539	0.0082	0.0255
	DenseNet	0.9463	0.9492	**0.0029**	669	6.7%	0.9584	0.0121	0.0337	0.9456	0.0064	0.0136
MNIST	Lenet1	0.9754	0.9935	0.0181	306	3.1%	0.9889	**0.0135**	0.0931	0.9856	0.0112	0.0325
	Lenet4	0.9848	0.9799	**0.0049**	298	3.0%	0.9906	0.0058	0.0542	0.9884	0.0057	0.0337
	Lenet5	0.9858	0.9960	0.0102	250	2.5%	0.9944	**0.0086**	0.0689	0.9909	0.0068	0.0403
CIFAR-100	NIN	0.5698	0.5625	**0.0073**	1,360	13.6%	0.6031	0.0333	0.0097	0.5700	0.0081	0.0061
	VGG19	0.6159	0.6082	**0.0077**	1,294	12.9%	0.6525	0.0366	0.0080	0.6175	0.0097	0.0064
	ResNet50	0.5541	0.5609	**0.0068**	1,797	17.9%	0.5902	0.0293	0.0102	0.5536	0.0098	0.0044
	GoogleNet	0.6939	0.6966	**0.0027**	1,447	14.5%	0.7504	0.0565	0.0126	0.6904	0.0086	0.0056
IMAGENET	ResNet50	0.7470	0.7457	**0.0013**	4239	8.5%	0.7754	0.0284	0.0070	0.7483	0.0052	0.0021
	VGG16	0.7127	0.7087	**0.0040**	3985	8.0%	0.7246	0.0119	0.0049	0.7112	0.0060	0.0022
	VGG19	0.7126	0.7089	**0.0037**	4029	8.1%	0.7268	0.0142	0.0048	0.7112	0.0054	0.0023
Average		0.8127	0.8121	0.0062	1,407	7.5%	0.8300	0.0215	0.0334	0.8136	0.0085	0.0174

TABLE III: Model information and the effectiveness of DeepReduce (using the same termination criterion).

Dataset	Model	Original Acc	DeepReduce				CES_{kl}			RANDOM_{kl}		
			Acc	Δacc	Size	Ratio	Acc	Δacc	Ratio	Acc	Δacc	Ratio
CIFAR-10	NIN	0.8815	0.8754	0.0061	289	2.9%	0.8793	0.0022	29.4%	0.8819	0.0026	47.8%
	VGG19	0.9346	0.9249	0.0097	493	4.9%	0.9336	0.0010	33.0%	0.9351	0.0019	48.8%
	ResNet	0.9220	0.9194	0.0026	273	2.7%	0.9228	0.0008	18.5%	0.9213	0.0021	48.4%
	WideResNet	0.9537	0.9491	0.0046	373	3.7%	0.9533	0.0004	33.2%	0.9535	0.0019	48.3%
	DenseNet	0.9463	0.9492	0.0029	669	6.7%	0.9463	0.0000	34.2%	0.9460	0.0019	48.2%
MNIST	Lenet1	0.9754	0.9935	0.0181	306	3.1%	0.9861	0.0107	35.4%	0.9857	0.0103	48.9%
	Lenet4	0.9848	0.9799	0.0049	298	3.0%	0.9887	0.0039	18.2%	0.9890	0.0042	48.2%
	Lenet5	0.9858	0.9960	0.0102	250	2.5%	0.9910	0.0052	17.9%	0.9907	0.0049	49.2%
CIFAR-100	NIN	0.5698	0.5625	0.0073	1,360	13.6%	0.5778	0.0080	32.7%	0.5691	0.0040	48.6%
	VGG19	0.6159	0.6082	0.0077	1,294	12.9%	0.6189	0.0030	32.8%	0.6159	0.0037	48.5%
	ResNet50	0.5541	0.5609	0.0068	1,797	17.9%	0.5626	0.0085	48.1%	0.5541	0.0034	48.8%
	GoogleNet	0.6939	0.6966	0.0027	1,447	14.5%	0.6959	0.0020	45.1%	0.6941	0.0034	48.9%
IMAGENET	ResNet50	0.7470	0.7457	0.0013	4239	8.5%	0.7703	0.0233	23.8%	0.7466	0.0034	80.2%
	VGG16	0.7127	0.7087	0.0040	3985	8.0%	0.7128	0.0001	21.6%	0.7138	0.0037	79.1%
	VGG19	0.7129	0.7089	0.0037	4029	8.1%	0.7113	0.0013	21.6%	0.7122	0.0038	79.1%
Average		0.8127	0.8121	0.0062	1,407	7.5%	0.8167	0.0047	29.7%	0.8139	0.0037	54.7%

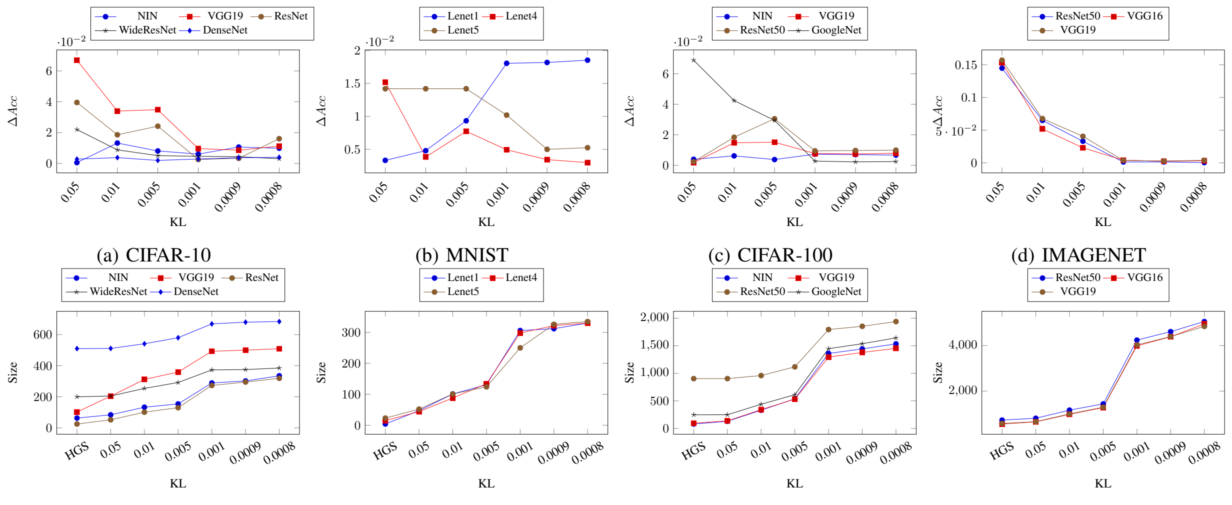
early estimation on different datasets may varies a lot, which is also verified according our evaluation results, the performance of DeepReduce is similar on different datasets in terms of ΔAcc. However, the performance of CES varies a lot. For example, the ΔAcc of CES on CIFAR100 is much larger than the one on other datasets.

In summary, the results show that DeepReduce can save over 92.5% (1-7.5%) testing cost on average. Besides, the selected testing data of DeepReduce achieve a similar test accuracy as the whole testing data do, indicating that DeepReduce is effective in performance estimation of DL models.

2) Comparison using the same KL value: Table III presents the comparison results of DeepReduce, CES_{kl}, and RANDOM_{kl}, each of which uses the same KL value (i.e., 0.001) in the termination criterion. DeepReduce outperforms CES_{kl} and RANDOM_{kl}in terms of the number of selected testing data. Note that all the three approaches, i.e., DeepReduce, CES_{kl}, and RANDOM_{kl}, use the same setting of α, but DeepReduce selects much less testing data compared with the others. From the table, the amount of selected data obtained by CES_{kl} range from 17.9% to 48.1%, with an average of 29.7%, while the amount of selected data obtained by random approach range from 47.8% to 80.2%, with an average of 54.7%. Thus, DeepReduce outperforms the other approaches on average in reduction size, while providing similar performance estimation.

CES and CES_{kl} are random sampling approaches with a carefully designed strategy, and thus they may not always perform well. In other words, it is not clear whether CES/CES_{kl} is able to achieve consistently good performance in practice. To answer this question, we run CES on ResNet (CIFAR-10) for 50 times, and find that the test accuracy achieved by the selected testing data ranges from 0.9011 to 0.9744, indicating that the performance of CES is not stable in practice. In comparison, DeepReduce is a stable approach without uncertainty, indicating that DeepReduce is more stable than CES/CES_{kl}. Furthermore, CES/CES_{kl} uses the output distribution of neurons in the last hidden layer to guide sampling. Considering the large number of neurons in the last hidden layer, the time cost of CES/CES_{kl} tends to be large. For example, on ResNet50 (CIFAR-100), CES spends more than 3 hours on testing data reduction and achieves 17.9% data reduction with ΔAcc of 0.0379 and KL value of 0.0102. In comparison, DeepReduce spends 376 seconds to reduce the testing data to 17.9% and achieves the ΔAcc of 0.0094 and KL of 0.0010. Thus, DeepReduce is more practical than CES/CES_{kl} as the former is more stable and efficient.

Besides, DeepReduce selects more testing data on two datasets (i.e., CIFAR-100 and IMAGENET) than the others (i.e., CIFAR-10 and MNIST), and we suspect the reason to be the differences between datasets. In particular, both CIFAR-10 and MNIST divide all images into 10 classes, while CIFAR-


(a) CIFAR-10 (b) MNIST (c) CIFAR-100 (d) IMAGENET

(e) CIFAR-10 (f) MNIST (g) CIFAR-100 (h) IMAGENET

Fig. 1: Results with different termination criteria (α) on four datasets.

TABLE IV: Results of DeepReduce with various neuron coverage criteria when α in KL is set to be 0.001.

Dataset	Model	NC(0.25)		NC(0.5)		NC(0.75)		NBC		SNAC		TKNC(1)		TKNC(2)		TKNC(3)	
		ΔAcc	Size	ΔAcc	Size	ΔAcc	Size	ΔAcc	Size	ΔAcc	Size	ΔAcc	Size	ΔAcc	Size	ΔAcc	Size
CIFAR-10	NIN	0.0119	276	0.0061	289	0.0074	369	0.0057	532	0.0081	498	0.0045	512	0.0018	482	0.0267	434
	VGG19	0.0130	286	0.0097	493	0.0094	3288	0.0034	1886	0.0094	1358	0.0068	4545	0.0071	4717	0.0044	4654
	ResNet	0.0021	287	0.0026	273	0.0307	275	0.0118	619	0.0070	459	0.0007	608	0.0024	485	0.0005	426
	WideResNet	0.0133	302	0.0046	373	0.0021	1493	0.0029	2257	0.0005	1377	0.0115	2925	0.0074	2793	0.0064	2886
	DenseNet	0.0106	325	0.0029	669	0.0097	1261	0.0021	967	0.0044	671	0.0053	1458	0.0041	1291	0.0002	1168
MNIST	Lenet1	0.0046	274	0.0181	306	0.0173	275	0.0075	293	0.0061	270	0.0073	289	0.0147	302	0.0117	310
	Lenet4	0.0037	317	0.0049	298	0.0047	285	0.0017	296	0.0043	275	0.0020	304	0.0079	275	0.0080	279
	Lenet5	0.0078	313	0.0102	250	0.0108	298	0.0016	318	0.0106	279	0.0045	308	0.0014	320	0.0003	276
CIFAR-100	NIN	0.0011	1368	0.0073	1360	0.0048	1354	0.0029	1547	0.0076	1414	0.0010	1540	0.0037	1388	0.0195	136
	VGG19	0.0186	1346	0.0077	1294	0.0033	1376	0.0011	2209	0.0087	1710	0.0052	2394	0.0108	2191	0.0057	2035
	ResNet50	0.0062	1326	0.0068	1797	0.0081	3248	0.0030	4906	0.0118	3520	0.0031	3647	0.0083	3647	0.0048	3625
	GoogleNet	0.0176	1338	0.0027	1447	0.0083	1759	0.0024	2065	0.0015	1681	0.0014	2580	0.0041	2347	0.0085	2228

100 and IMAGENET divide all images into 100 and 1,000 classes, respectively. Therefore, on the latter two datasets, more testing data are needed to be selected to be representative of a larger number of classes.

To sum up, DeepReduce accurately estimates the performance of DL models with less testing data. Furthermore, it outperforms the baseline approaches and is more practical.

B. Results for RQ2

We investigated the performance of our approach with different independent variables (i.e., neuron coverage criteria and termination criteria). We set α in the termination criterion to different values, i.e., 0.05, 0.01, 0.005, and 0.001, and use various coverage criteria. To understand the influence caused by termination criteria, we applied DeepReduce with various α values but fixed coverage criterion NC(0.5), and the results are shown in Fig.1. In these figures, each line represents the results of one model.

From these figures, when α is smaller than 0.001, the changes in ΔAcc produced by the selected testing data are small. When α is larger than 0.001, the ΔAcc fluctuates with different α values. On the other hand, the smaller the α is, the more the testing data are selected by DeepReduce. Moreover, when α is smaller than 0.001, much more testing data are needed in order to satisfy the requirement of α. Considering both ΔAcc and reduction size, $\alpha = 0.001$ is a practical choice.

In Fig.1e-1h, the "HGS" in x-axis represents the number of testing data selected in the first phase (HGS). When setting $\alpha = 0.001$ in the algorithm, DeepReduce selects more testing data in the second phase than the data selected in the first phase, with the purpose of approximating output distribution. That is, the testing data selected for the purpose of testing adequacy cannot satisfy the requirement on similar output distribution. Therefore, more testing data are needed in order to achieve the similar output distribution after the first phase (with HGS). This observation can be found on all the datasets.

To figure out the influence caused by various neuron coverage criteria, we set α in the termination criterion to 0.001 and investigate the performance of DeepReduce with various coverage criteria. The results are shown in Table IV. From the table, under different neuron coverage criteria, the ΔAcc achieved by DeepReduce in estimating DL model performance is close. However, different coverage criteria influence the number of selected testing data to a certain extent. NC(0.5) and NC(0.25) tend to select few testing data, whereas three TKNC tend to select much more testing data. For example, on VGG19 (CIFAR10), DeepReduce selects 4,545 testing data when using TKNC(1), which is much larger than the number obtained by other criteria (e.g., NC(0.5) and NBC). Through further analysis, we found that the large number of selected testing data are caused by the first phase of our reduction algorithm. As the testing adequacy measured by TKNC is hard to satisfy,

TABLE V: Sensitivity results for different components

Model	Variant$_1$		Variant$_2$		Variant$_3$		Variant$_4$	
Metrics	ΔAcc	KL	ΔAcc	KL	ΔAcc	KL	ΔAcc	KL
NIN	0.0147	0.001	0.0095	0.017	0.0684	0.017	0.0147	0.031
VGG19	0.0045	<0.001	0.0025	0.011	0.0928	0.028	0.0157	0.033
ResNet	0.0157	0.001	0.0209	0.010	0.1015	0.069	0.0062	0.039
WideResNet	0.0180	0.001	0.0020	0.019	0.0985	0.039	0.0007	0.025
DenseNet	0.0059	<0.001	0.0091	0.012	0.0450	0.010	0.0031	0.017
Average	0.0118	0.001	0.0088	0.014	0.0634	0.033	0.0080	0.029

TABLE VI: Effectiveness of DeepReduce in regression scenarios (on the NIN model)

Metrics	Modifications						
	Structural Changes				Non-Structural Changes		
	ADL	DEL	ADN	DEN	LR	MO	DR
Acc(Actual)	0.8816	0.8734	0.8857	0.8808	0.8480	0.8699	0.8820
Acc(DeepReduce)	0.8731	0.8625	0.8776	0.8676	0.8350	0.8600	0.8715
ΔAcc	0.0085	0.0108	0.0089	0.0131	0.0130	0.0099	0.0105

DeepReduce tends to select more testing data to achieve the same neuron coverage. For example, on VGG19 (CIFAR10), DeepReduce selects 1,655 testing data by adopting HGS on TKNC(1), whereas selects 94 testing data by adopting HGS on NC(0.5). However, the similarity of the output distribution between the testing data selected by HGS and the original ones is not small. In order to approximate the output distribution between the selected testing data and the original ones, a great amount number of testing data are needed in the second phase of DeepReduce. Similar observation can also be found on the results using other neuron coverage criteria. In sum, neuron coverage criteria affect slightly on DeepReduce in terms of accuracy, but have more important influence on DeepReduce in terms of the number of selected testing data.

Note that different neuron coverage criteria are designed with different purposes. For example, NC is designed to measure the degree of neuron activation, while NBC is designed to measure to what extent the corner-case region is covered by testing data. DeepReduce is not specific to one criterion, but general to various criteria. Therefore, in the future we will investigate the performance of DeepReduce with more criteria.

In summary, in terms of estimation accuracy, DeepReduce tends to perform better when the termination criterion gets stricter (a smaller α), and it achieves similar performance with different neuron coverage. In our experiments, we set $\alpha = 0.001$ and using NC(0.5) as default settings.

C. Results for RQ3

The sensitivity results are given by Table V. The comparison results of DeepReduce can refer to Column "DeepReduce" in Table II, while the KL values of DeepReduce are always 0.001. In general, the ΔAcc results and KL values of each variant are usually larger than DeepReduce, indicating that each component positively contributes to the performance of DeepReduce. In other words, without either component, DeepReduce tends to perform worse.

In particular, in terms of ΔAcc, Variant$_3$ performs the worst comparing with the other variants, indicating that the corresponding component (i.e., the division strategy for output distribution) contributes the most for DeepReduce. It is reasonable as the division strategy performs well in distinguishing the distribution of the outputs of neurons in the last layer. Each neuron in the last layer represents the probability of being divided into a particular class. Given a neuron, those testing data which do not fall into this class will be assigned with very small values (e.g., less than 0.1). However, only a small amount of testing data will be classified into the same class, while many other testing data do not fall into this class. That is, the output distribution on each neuron is more likely heavy-tailed. The division strategy used in DeepReduce divide the output range into several sections, each of which contains the same number of unique output. Therefore, it performs well in distinguishing the heavy-tailed distribution.

In terms of KL values, all the components except the first phase contribute to the performance of DeepReduce. Note that a small KL value indicates that the similarity between the selected testing data and the original ones is large. From the table, the KL values achieved by Variant$_2$, Variant$_3$, and Variant$_4$ are much larger than 0.001, which is the recommended KL value according to our evaluation results presented in Section IV-B. That is, more testing data need to be selected by these variants in order to achieve a similar output distribution. Considering that much more testing data are selected to reduce the KL values (mentioned in Section IV-B), these three components also contribute to the reduction size.

In addition, the time overhead of Variant$_2$ and Variant$_3$ are larger than the others including DeepReduce. For example, on VGG19 (CIFAR-10), the time overhead of Variant$_2$ and Variant$_3$ are 1,414 and 563 seconds, while the time overhead of DeepReduce is 14 seconds. For Variant$_2$, the number of neurons in the last-hidden layer is much larger than the ones in the last layer, and thus DeepReduce is more time-consuming. For Variant$_3$, improper division strategy tends to place a great proportion of data (e.g., about 90%) in the same section, and thus KL minimization (Lines 29-33 in Algorithm 1) is more time-consuming.

D. Results for RQ4

Due to space limit, we present only the average testing results on the NIN (CIFAR-10) in Table VI, where Row "Acc(Actual)" presents the actual accuracy achieved by the whole testing data, Row "Acc(DeepReduce)" presents the accuracy achieved by the testing data selected by DeepReduce, and Row "ΔAcc" presents the difference between the above two values. According to the table, the average ΔAcc ranges from 0.0085 to 0.0131, indicating that DeepReduce performs still well in estimating the performance of DL models even when they have been slightly modified. However, compared with Table II, the average ΔAcc of DeepReduceincreases from 0.0070 to 0.0024, indicating the performance of DeepReduce in regression testing is a bit worse, resulting from model modification.

Among the seven types of modifications, the ΔAcc values of DEN and LR are the largest, while those of ADL and ADN are the smallest, indicating that the former two types of modification harm the performance of DeepReduce the most, whereas ADL and ADN harms the performance of

DeepReduce the least. Besides, we find that DeepReduce is more stable with various MO changes by further analysis.

Note that this experiment is conducted in a simplified regression testing scenario, where the model is changed slightly by following the given types of modification. In the future, we will investigate the performance of DeepReduce with more complex modifications.

V. Related Work

A. Cost of Deep Learning Testing

Most of the existing work on DL testing focuses on testing data generation [7], [13], [27]–[29]. However, due to the large number of testing data, DL testing often suffers from the cost problem. For example, a large number of testing data require much efforts on manual labelling, and thus some work investigates how to reduce this cost. In particular, Li et al. [9] proposed two approaches CSS and CES to reduce the amount of testing data required to be labelled based on cross-entropy minimization. Chen et al. [30] proposed a cluster-based selection technique to reduce labeling cost. Recently, Shi et al. [31] proposed a prioritization approach DeepGiNi, which identifies the testing data to be labelled earlier based on the gini impurity of testing data. These works are related to ours, because in general all of us target at the cost problem in DL testing. However, these approaches address the cost problem in different ways (i.e., selection, reduction or prioritization using different algorithms with slightly different objectives). Besides, DeepReduce is more stable on various models and its improvement over CES is about 71.2% on average according to the experiment results.

B. Test Reduction

Test reduction, also called as test minimization, aims to obtain the minimal subset of the original test suite by satisfyng the original test requirements [15], [32], [33]. In the literature, a large number of test reduction algorithms have been proposed. For example, Harrold et al. [15] proposed a greedy heuristic (HGS) that repeatedly selects tests from the remaining tests with minimal cardinality. Chen and Lau [32] proposed another heuristic that identified essential and 1-to-1 redundant tests from the original ones, which iteratively picks essential, removes 1-to-1 redundant and greedily selects tests covering a maximal number of unsatisfied requirements. Black et al. [34] formulated the problem as a binary ILP model by encoding the def-use coverage criteria and fault-revealing ability of tests into the objective function. Besides the algorithm, various conventional criteria have been proposed and used to guide test reduction, e.g., structural coverage [35], fault detection [36], [37], energy consumption [38], [39], and even hybrid combinations of different coverage [40]. Our work is similar to test reduction because both of them are to set a subset of test cases (or testing data). However, our work is the first one to early estimate a DL model's performance, and thus the selection criteria are different and the algorithms proposed in conventional test reduction cannot be used directly to solve our problem.

C. Criteria in DL Testing

To measure the testing adequacy in DL testing, various criteria have been proposed in the literature, and have been used in testing data generation [3], [5], [13], [41]. The mostly studied testing adequacy criteria are neuron coverage criteria [13], [14], [42]. Besides the neuron coverage criteria used or mentioned in prior sections, some other neuron coverage criteria also exist. For example, Sun et al. [42] proposed a family of fine-grained MC/DC neuron coverage criteria, Ma et al. [43] proposed neuron state combination based criteria, and recently Du et al. [44] proposed five coverage criteria for RNNs besides two trace similarity metrics, including three state-level and two transition-level coverage criteria. In addition to neuron coverage criteria, there are also some other types of criteria. In particular, Ma et al. [45] proposed a mutation-based criterion for DNN testing by designing mutation operators for DNNs. Besides, Islam et al. [46] also classified some modifications for DNNs, which may be viewed as mutation operators. Kim et al. [47] proposed surprise adequacy for DNNs, which measures the dissimilarity between a test and the training data set. Gerasimou et al. [48] proposed an Importance-Driven testing adequacy criterion (IDC) for DL systems. In this paper, we use some neuron coverage criteria to be representative of testing adequacy in the evaluation, but our approach is not specific to any specific criteria.

VI. Conclusion

In this paper, we defined an input reduction problem for early estimation of DL model performance with the purpose of cost-effective testing, and then proposed a two-phase approach DeepReduce, which selects a subset of testing data by satisfying both testing adequacy and output distribution similarity. To evaluate the proposed approach, we conducted an experimental study on 15 DL models with four datasets. On average, DeepReduce reduces the whole testing data to 7.5% with the accuracy loss smaller than 0.0062. We conducted sensitivity analysis on the components of DeepReduce and found that each of them has contributions to the performance of DeepReduce. Moreover, in the simplified regression testing scenario given in this paper, the average accuracy loss in regression testing is only 0.0104, indicating that DeepReduce may be still useful for models with slight modification.

Acknowledgment

This work is partially supported National Natural Science Foundation of China under Grant Nos. 61872008, 61861130363, and 61828201, as well as Australian Research Council (ARC) Discovery Project DP200102940.

References

[1] E. Amir, "Uber finds deadly accident likely caused by software set to ignore objects on road." *The information*, 2018.

[2] R. Zhang, W. Xiao, H. Zhang, Y. Liu, H. Lin, and M. Yang, "An empirical study on program failures of deep learning jobs," in *Proceedings of the 42nd International Conference on Software Engineering (ICSE '20)*. IEEE/ACM, July 2020.

[3] X. Xie, L. Ma, F. Juefei-Xu, M. Xue, H. Chen, Y. Liu, J. Zhao, B. Li, J. Yin, and S. See, "Deephunter: a coverage-guided fuzz testing framework for deep neural networks," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2019, Beijing, China, July 15-19, 2019*. ACM, 2019, pp. 146–157.

[4] Z. Wang, M. Yan, J. Chen, S. Liu, and D. Zhang, "Deep learning library testing via effective model generation," in *The 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, to appear.

[5] Y. Sun, M. Wu, W. Ruan, X. Huang, M. Kwiatkowska, and D. Kroening, "Concolic testing for deep neural networks," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 2018, pp. 109–119.

[6] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *CVPR'14*, June 2014, pp. 1701–1708.

[7] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*. ACM, 2018, pp. 303–314.

[8] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, Dec 2015.

[9] Z. Li, X. Ma, C. Xu, C. Cao, J. Xu, and J. Lü, "Boosting operational DNN testing efficiency through conditioning," in *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*. ACM, 2019, pp. 499–509.

[10] A. L. Gibbs and F. E. Su, "On choosing and bounding probability metrics," *International statistical review*, vol. 70, no. 3, pp. 419–435, 2002.

[11] H. Wang, J. Xu, C. Xu, X. Ma, and J. Lu, "Dissector: Input validation for deep learning applications by crossing-layer dissection," in *Proceedings of the 42nd International Conference on Software Engineering (ICSE '20)*. IEEE/ACM, July 2020.

[12] S. Ma, Y. Liu, W. Lee, X. Zhang, and A. Grama, "MODE: automated neural network model debugging via state differential analysis and input selection," in *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*. ACM, 2018, pp. 175–186.

[13] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *SOSP'17*. ACM, 2017, pp. 1–18.

[14] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu *et al.*, "Deepgauge: Multi-granularity testing criteria for deep learning systems," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*. ACM, 2018, pp. 120–131.

[15] M. J. Harrold, R. Gupta, and M. L. Soffa, "A methodology for controlling the size of a test suite," *ACM TOSEM*, vol. 2, no. 3, pp. 270–285, 1993.

[16] S. Kullback and R. A. Leibler, "On information and sufficiency," *Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[17] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[18] A. Krizhevsky, V. Nair, and G. Hinton, "The cifar dataset," *online: http://www.cs.toronto. edu/kriz/cifar.html*, vol. 55, 2014.

[19] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li, "Imagenet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015.

[20] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[21] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR'16*, 2016, pp. 770–778.

[23] S. Zagoruyko and N. Komodakis, "Wide residual networks," *CoRR*, vol. abs/1605.07146, 2016.

[24] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *CoRR*, vol. abs/1608.06993, 2016.

[25] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR'15*. IEEE Computer Society, 2015, pp. 1–9. [Online]. Available: https://doi.org/10.1109/CVPR.2015.7298594

[26] F. Wilcoxon, "Individual comparisons by ranking methods," in *Breakthroughs in statistics*. Springer, 1992, pp. 196–202.

[27] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*. ACM, 2018, pp. 132–142.

[28] X. Gao, R. K. Saha, M. R. Prasad, and A. Roychoudhury, "Fuzz testing based data augmentation to improve robustness of deep neural networks," in *Proceedings of the 42nd International Conference on Software Engineering, ICSE 2020, Seoul, South Korea, July 2020*. IEEE / ACM, 2020.

[29] X. Zhang, X. Xie, L. Ma, X. Du, Q. Hu, Y. Liu, J. Zhao, and M. Sun, "Towards characterizing adversarial defects of deep learning software from the lens of uncertainty," in *Proceedings of the 42nd International Conference on Software Engineering, ICSE 2020, Seoul, South Korea, July 2020*. IEEE / ACM, 2020.

[30] J. Chen, Z. Wu, Z. Wang, H. You, L. Zhang, and M. Yan, "Practical accuracy estimation for efficient dnn testing," *ACM Transactions on Software Engineering and Methodology*, 2020, to appear.

[31] Q. Shi, J. Wan, Y. Feng, C. Fang, and Z. Chen, "Deepgini: Prioritizing massive tests to reduce labeling cost," *CoRR*, vol. abs/1903.00661, 2019.

[32] T. Y. Chen and M. F. Lau, "Dividing strategies for the optimization of a test suite," *Inf. Process. Lett.*, vol. 60, no. 3, pp. 135–141, 1996.

[33] D. Hao, L. Zhang, X. Wu, H. Mei, and G. Rothermel, "On-demand test suite reduction," in *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 2012, pp. 738–748.

[34] J. Black, E. Melachrinoudis, and D. R. Kaeli, "Bi-criteria models for all-uses test suite reduction," in *26th International Conference on Software Engineering (ICSE 2004), 23-28 May 2004, Edinburgh, United Kingdom*, A. Finkelstein, J. Estublier, and D. S. Rosenblum, Eds. IEEE Computer Society, 2004, pp. 106–115.

[35] J. Chen, Y. Bai, D. Hao, L. Zhang, L. Zhang, and B. Xie, "How do assertions impact coverage-based test-suite reduction?" in *2017 IEEE International Conference on Software Testing, Verification and Validation, ICST 2017, Tokyo, Japan, March 13-17, 2017*. IEEE Computer Society, 2017, pp. 418–423.

[36] M. A. Alipour, A. Shi, R. Gopinath, D. Marinov, and A. Groce, "Evaluating non-adequate test-case reduction," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016*. ACM, 2016, pp. 16–26.

[37] D. Hao, L. Zhang, X. Wu, H. Mei, and G. Rothermel, "On-demand test suite reduction," in *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*. IEEE Computer Society, 2012, pp. 738–748.

[38] R. J. Behrouz, A. Sadeghi, H. Bagheri, and S. Malek, "Energy-aware test-suite minimization for android apps," in *Proceedings of the 25th International Symposium on Software Testing and Analysis, ISSTA 2016, Saarbrücken, Germany, July 18-20, 2016*. ACM, 2016, pp. 425–436.

[39] D. Li, Y. Jin, C. Sahin, J. Clause, and W. G. J. Halfond, "Integrated energy-directed test suite optimization," in *International Symposium on Software Testing and Analysis, ISSTA '14, San Jose, CA, USA - July 21 - 26, 2014*. ACM, 2014, pp. 339–350.

[40] S. Sampath, R. C. Bryce, and A. M. Memon, "A uniform representation of hybrid criteria for regression testing," *IEEE Trans. Software Eng.*, vol. 39, no. 10, pp. 1326–1344, 2013.

[41] A. Odena, C. Olsson, D. Andersen, and I. J. Goodfellow, "Tensorfuzz: Debugging neural networks with coverage-guided fuzzing," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, ser. Proceedings of Machine Learning Research, vol. 97. PMLR, 2019, pp. 4901–4911.

[42] Y. Sun, X. Huang, and D. Kroening, "Testing deep neural networks," *CoRR*, vol. abs/1803.04792, 2018. [Online]. Available: http://arxiv.org/abs/1803.04792

[43] L. Ma, F. Juefei-Xu, M. Xue, B. Li, L. Li, Y. Liu, and J. Zhao, “Deepct: Tomographic combinatorial testing for deep learning systems,” in *26th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2019, Hangzhou, China, February 24-27, 2019*. IEEE, 2019, pp. 614–618.

[44] X. Du, X. Xie, Y. Li, L. Ma, Y. Liu, and J. Zhao, “Deepstellar: model-based quantitative analysis of stateful deep learning systems,” in *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*. ACM, 2019, pp. 477–487.

[45] L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. Juefei-Xu, C. Xie, L. Li, Y. Liu, J. Zhao, and Y. Wang, “Deepmutation: Mutation testing of deep learning systems,” in *29th IEEE International Symposium on Software Reliability Engineering, ISSRE 2018, Memphis, TN, USA, October 15-18, 2018*. IEEE Computer Society, 2018, pp. 100–111.

[46] M. J. Islam, R. Pan, G. Nguyen, and H. Rajan, “Repairing deep neural networks: Fix patterns and challenges,” in *Proceedings of the 42nd International Conference on Software Engineering (ICSE ’20)*. IEEE/ACM, July 2020.

[47] J. Kim, R. Feldt, and S. Yoo, “Guiding deep learning system testing using surprise adequacy,” in *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*. IEEE / ACM, 2019, pp. 1039–1049.

[48] S. Gerasimou, H. F. Eniser, A. Sen, and A. Cakan, “Importance-driven deep learning system testing,” in *Proceedings of the 42nd International Conference on Software Engineering, ICSE 2020, Seoul, South Korea, July 2020*. IEEE / ACM, 2020.