DeepAbstraction: 2-Level Prioritization for Unlabeled Test Inputs in Deep Neural Networks

Hamzah Al Qadasi, Changshun Wu, Yliès Falcone, Saddek Bensalem

DeepAbstraction: 2-Level Prioritization for Unlabeled Test Inputs in Deep Neural Networks

Hamzah Al-Qadasi[1], Changshun Wu[1], Yliès Falcone[2], Saddek Bensalem[1]
[1]*Univ. Grenoble Alpes, CNRS, Grenoble INP, Verimag, 38000 Grenoble, France*
[2]*Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, 38000 Grenoble, France*
{hamzah.al-qadasi, changshun.wu, ylies.falcone, saddek.bensalem}@univ-grenoble-alpes.fr

***Abstract*—Deep learning systems recently achieved unprecedented success in various industries. However, DNNs still exhibit some erroneous behaviors, which lead to catastrophic results. As a result, more data should be collected to cover more corner cases. On the other hand, a massive amount of data consumes more human annotators (oracle), which increases the labeling budget and time. We propose an effective test prioritization technique, called *DeepAbstraction* to prioritize the more likely error-exposing instances among the entire unlabeled test dataset. The ultimate goal of our framework is to reduce the labeling cost and select the potential corner cases earlier before production. Different from existing work, DeepAbstraction leverages runtime monitors. In the literature, runtime monitors are primarily used to supervise the prediction of the neural network. Then, monitors trigger a verdict for each prediction: acceptance, rejection, or uncertainty. Monitors quantify the acquired knowledge into box abstraction during the training. Each box abstraction contains instances that share similar high-level features. In the test part, the verdict of monitor depends in which box abstraction a test instance resides. Moreover, we study intensively where corner cases can reside in the feature space, either near-boundary regions or near-centroid regions. The existing test prioritization techniques can only prioritize many near-boundary instances and a few near-centroid instances. Nevertheless, DeepAbstraction can effectively prioritize numerous instances from both regions. Therefore, our evaluation shows that DeepAbstraction outperforms the state-of-the-art test prioritization techniques.**

***Index Terms*—Test Prioritization, Big Data, Deep Learning Testing, Runtime Monitoring, Effective Labeling.**

I. Introduction

Data become rapidly more and more massive than ever to empower the learning capability of deep neural networks, e.g., a large-scale person Re-ID dataset contains more than 29M footprints [1]. The labeling process can be a time-consuming and labor-intensive problem in supervised learning. The problem becomes worse in two situations: first, labeling in some domains requires deep knowledge and high experience, e.g., medical and military fields; second, labeling in multi-class segmentation when an image contains 2, 3, or more classes. Both training and test data require the labeling process. In the training data, active learning (AL) techniques are mainly used to reduce the burden of labeling costs. For test data, a typical way is to use *test prioritization* techniques that estimate the error-revealing capability of unlabeled test instances to expose the misbehavior of the trained neural network.

There are a few recently published works in test prioritization for deep learning systems, such as Byun et al. [2], DeepGini [3], PRIMA [4], and TestRank [5]. However, these techniques suffer from performance issues. For instance, [2] and [3] use the classifier uncertainty to estimate the error-revealing capability of test instances, however, it is a non-robust distance-based metric [6]. Sec. VI discusses the limitations of other test prioritization techniques in more details.

Usually, the classifier uncertainty is measured by the distance between an instance and the decision boundary [7], i.e., the near-boundary instances show higher uncertainty and are much more likely to be misclassified than near-centroid instances [8]. However, in practice, the neural networks classify some near-centroid instances incorrectly with very high confidence. The root explanation is twofold: first, misclassified near-centroid instances can hold similar features to the surrounding correctly classified instances in the feature space. However, the ground truths of misclassified near-centroid instances belong to other classes, as shown in Fig. 1. Second, the classifier uncertainty is a distance-based metric that neglects the position of misclassified near-centroid instances, whether close to the centroid of ground truth or the centroid of the incorrectly classified class. As a result, there is a pressing need to prioritize misclassified instances near the centroids and decision boundaries. Nonetheless, it is challenging to identify misclassified near-centroid instances because correctly classified instances surround them.

In this paper, we demonstrate the effectiveness of runtime monitors [9], [10] in identifying misclassified instances in both regions. Monitors can generalize the knowledge acquired during the training through the so-called box abstraction (Sec. II-B1). We construct box abstraction for both correctly and incorrectly classified instances. More specifically, there are two types of boxes according to the misclassified instances: boxes for near-centroid instances and boxes for near-boundary ones. The former separates the misclassified instances from the surrounding correctly classified instances in the feature space. Thus, during testing, monitors reject the network prediction for those located inside the near-centroid boxes.

More interestingly, monitors do not rely on the uncertainty metric; thus, they significantly outperform existing approaches, as demonstrated in (Sec. V). On several benchmarks, Deep-

This paper is supported by the European project Horizon 2020 research and innovation programme under grant agreement No. 956123 and by the French National Research Agency (ANR) in the framework of the Investissements d'Avenir program (ANR-10-AIRT-05, irtnanoelec).

(a) labeled: 0 Predicted: 6 (b) labeled: 2 Predicted: 7 (c) labeled: 6 Predicted: 0 (d) labeled: 7 Predicted: 2

Fig. 1: Misclassified near-centroid instances are very close to the centroids of predicted classes.

Abstraction shows more stable and better performance than the state-of-the-art tool (TestRank). Moreover, contrarily to TestRank, our framework does not require any prior labeling for test instances to operate. In summary, our significant contributions are as follows:

- We conduct the first study investigating the effectiveness of monitors in the test prioritization area for deep learning systems.
- We introduce a comprehensive study for the regions of the misclassified instances and the regions where the neural network can misclassify the instances with high confidence.
- We achieve the state-of-the-art performance, demonstrated by empirically comparing our framework with other test prioritization techniques.

The rest of the paper is structured as follows: Sec. II provides essential background to understand how to construct monitors and the mechanism of monitors. Sec. III introduces the problem formulation and then explains the DeepAbstraction algorithm. The main setup to conduct the experiments is detailed in Sec. IV. In Sec. V, we report on the evaluation of our framework in comparison with other test prioritization work. Sec. VI summarizes the state-of-the-art test prioritization techniques with the main limitations. In Sec. VII, we conclude and discuss future work.

II. Preliminaries

In this section, we review the basic definitions of neural networks (Sec. II-A), box-based monitors (Sec. II-B), and statistical scoring functions (Sec. II-C).

A. Deep Neural Network

Let $\mathrm{X_t}$ and $\mathrm{X_s}$ denote the training and test datasets, respectively, where $\mathrm{X_t} = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$, where m is the number of the training instances, x is a network input, and y is the ground truth or the actual class. Similarly, let $\mathrm{X_s} = \{x_1, x_2, ..., x_r\}$, where x is an unlabeled input and r is the number of test instances. Let $\mathcal{N}$ be a neural network, and $\mathcal{N}(x)$ be the network prediction. $\mathcal{N}$ consists of a set of layers such that $\mathrm{L} = \{\mathrm{L_k} \mid 1 \leq k \leq q\}$, where q is the total number of layers in the neural network. The number of neurons in a layer $\mathrm{L_k}$ is denoted by $|\mathrm{L_k}|$. In a multi-classification problem, the last layer $\mathrm{L_q}$ is a softmax layer, and $\mathrm{L_{q-1}}$ layer, namely the penultimate layer, is the layer before the softmax layer.

B. Runtime Monitoring

Recently, various runtime monitors [9], [10] have been proposed for supervising the decision of neural network. In this paper, we follow the framework in [10], three-verdict monitors supervise how the neural network predicts the inputs and judge the network prediction with acceptance, uncertainty, or rejection. In the following, we recall how to construct and execute a monitor. For more details, we refer to [9], [10].

1) Monitor Construction: After training, for each training instance, we collect the high-level features from the penultimate layer and the corresponding predicted class. Let $\mathbf{watch}(x, \mathrm{L_k})$ be a function that reads the output of $\mathrm{L_k}$. The output of $\mathbf{watch}(x, L_k)$ is a $|\mathrm{L_k}|$ dimension vector, denoted as $\vec{\mathbf{v}}$. We consider a training dataset as a set of subsets such that $\mathrm{X_t} = \{\mathrm{X_1}, \mathrm{X_2}, ..., \mathrm{X_g}\}$ where g is the total number of classes in a dataset. All instances in each subset have the same ground truth. We also have a corresponding $\mathrm{V_i}$ for each subset $\mathrm{X_i}$, where $\mathrm{V_i} = \{\mathbf{watch}(x, \mathrm{L_{q-1}}) = \vec{\mathbf{v}} \mid x \in \mathrm{X_i}\}$.

After high-level features collection, we partition $\mathrm{V_i}$ into two subsets $\mathrm{V_i^c}$, and $\mathrm{V_i^{inc}}$ based on whether a neural network correctly or incorrectly classifies the input, where $\mathrm{V_i^c}$, and $\mathrm{V_i^{inc}}$ are formally defined as follows:

$$\mathrm{V_i^c} = \{\mathbf{watch}(x, \mathrm{L_{q-1}}) = \vec{\mathbf{v}} \quad | \quad x \in \mathrm{X_i},\, \mathcal{N}(x) = y_i\}$$

$$\mathrm{V_i^{inc}} = \{\mathbf{watch}(x, \mathrm{L_{q-1}}) = \vec{\mathbf{v}} \quad | \quad x \in \mathrm{X_i},\, \mathcal{N}(x) \neq y_i\}$$

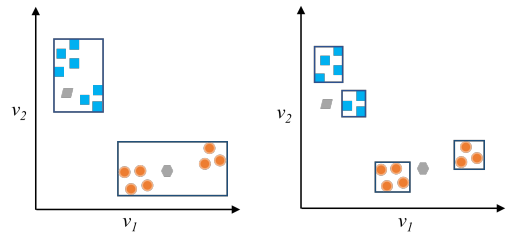
To construct box abstraction, assume $|\vec{\mathbf{v}}| = n$, $|\mathrm{V_i^c}| = p$, and $|\mathrm{V_i^{inc}}| = s$. Accordingly, there are two types of boxes $\mathrm{B_i^c}$ and, $\mathrm{B_i^{inc}}$ defined as follows:

$$B_i^c = \{[a_j, b_j] \mid 1 \leq j \leq n,\ a_j = \min_{k=1}^{p} V_i^c[j,k],\ b_j = \max_{k=1}^{p} V_i^c[j,k]\}$$

$$B_i^{inc} = \{[a_j, b_j] \mid 1 \leq j \leq n,\ a_j = \min_{k=1}^{s} V_i^{inc}[j,k],\ b_j = \max_{k=1}^{s} V_i^{inc}[j,k]\}$$

Box abstraction or *minimum bounding box* is a union of a list of intervals such that $\mathrm{B_i} = \{[a_1, b_1], [a_2, b_2], ..., [a_n, b_n]\}$ [10]. The underlying assumption is that instances of the same class show a similar pattern because they are more contiguous in the feature space than instances of other classes. Thus, monitors have two types of boxes for each class: one box contains correctly classified instances, and the other box contains incorrectly classified instances.

When a novel input is slightly similar to other instances inside the box, monitors falsely accept the network prediction. For example, in Fig. 2 (a), there are two classes of squares and circles, and the novel inputs are parallelogram and hexagon, respectively. The neural network incorrectly classifies the novel inputs as square and circle, respectively. We can obtain more accurate verdicts from monitors by clustering all instances into small boxes. In Fig. 2 (b), monitors correctly reject the predictions of the neural network for the novel inputs because the novel instances are outside the boxes. Therefore, clustering should proceed boxes construction. The clustering process has a hyperparameter τ, which controls the size of each box abstraction, i.e., the smaller the τ value is, the more compact the box abstraction is and vice versa.



(a) Without clustering (b) With clustering

Fig. 2: The monitor verdict to novel inputs before and after clustering [10].

2) Monitors Execution: During the evaluation of the neural network, if a test instance is inside one of the correctly classified boxes, the monitor accepts the prediction. If a test instance is inside an incorrectly classified box, the monitors reject the prediction. The verdict of monitors is uncertain if the test instance is in an overlapping region between the correctly and incorrectly classified boxes Lastly, if the test instance is outside all boxes, the monitors reject the network prediction. In the last case, monitors consider this instance a novel input or out-of-distribution(OOD).

C. Statistical Scoring Functions

To quantify the error-revealing capability of each test instance, we recall the definition of two scoring functions and how they will be used in test prioritization: Gini Impurity and entropy. They are heavily used in statistics [11] and machine learning fields [12], [13].

1) Gini Impurity or Gini Index: GI is a measure for the impurity of the distribution of classes over the output of the neural network. Gini Impurity is defined as:

$$GI = 1 - \sum_{i=1}^{C} p_i^2 \tag{1}$$

where p_i is the probability of an instance being classified to class i, C is the total number of classes, and $\sum_{i=1}^{C} p_i = 1$.

2) Entropy or Shanon entropy: entropy is a metric that measures the amount of randomness or variability in the distribution of classes over the network output. The following equation computes the value of entropy:

$$entropy = -\sum_{i=1}^{C} p_i \ \mathbf{log_2}(p_i) \tag{2}$$

III. Methodology

This section starts to define the problem formally and then provides the solution, i.e., the two-level prioritization technique. More specifically, we first explain the main regions where the misclassified instances reside in the feature space. Then, we describe the architecture of our framework, particularly the main components and functionalities. Lastly, we explain the logic and algorithm of DeepAbstraction.

A. Problem Formulation

Given a trained neural network $\mathcal{N}$ with a labeled training dataset X_t and an unlabeled test dataset X_s. We consider $|X_s|$ = n and the labeling cost is enough for only m test instances where $m \ll n$. We can utilize the test prioritization techniques to prioritize more error-exposing instances. Assuming there are k error-revealing instances in the test dataset and the cost of labeling $m = k$. Then, the ideal prioritization algorithm groups the test instances into two ordered sets as follows:

$$\mathrm{X_e} = \{F_s(x_i) \quad | \quad \mathcal{N}(x_i) \neq y_i, i \in [1,k], x \in \mathrm{X_s}\} \tag{3}$$

$$\mathrm{X_c} = \{F_s(x_i) \quad | \quad \mathcal{N}(x_i) = y_i, i \in [k+1,n], x \in \mathrm{X_s}\} \tag{4}$$

where $\mathrm{F_s}$ is a scoring function either Gini Impurity or entropy. Ideally, (3) shows that $\mathrm{X_e}$ contains all error-revealing test instances indexed between 1 and k. More specifically, for any $i, j \in [1,k]$ and $i < j$, then $\mathrm{F_s}(x_i) \geq \mathrm{F_s}(x_j)$. In addition, $\mathrm{F_s}(x_1)$ has the maximum score and $\mathrm{F_s}(x_k)$ has the smallest score in X_e. Eventually, X_e is further labeled by human annotators to evaluate a neural network. While (4) shows that $\mathrm{X_c}$ includes all correctly classified test instance. There is no cost left to label $\mathrm{X_c}$ due to $m = k$.

B. Algorithm

Our framework starts with training the neural network, as shown in Fig. 3. Then, it extracts the high-level features and groups the vectors based on the predicted class. These vectors contribute to building the boxes. During the test part, DeepAbstraction can be seen as a two-level ranking system. First, it groups the test instances into 3 groups based on the monitor verdict. Second, it prioritizes the instances within each group based on the value assigned by the scoring function. Finally, annotators label prioritized instances according to the labeling budget.

Next, we explain the main stages of our framework to perform test prioritization according to the algorithm 1.

1) Testing part: This part involves feeding the neural network with all instances in the unlabeled test dataset. After that, we index the instance to efficiently process the data by indices (line 3). Then, we extract the high-level features vector for each test input and read the corresponding predicted class y' (lines 4-5). The softmax function computes the classification probability of each class *output* (line 6). Next, we employ one of the two statistical scoring functions, either Gini Impurity or entropy, to calculate the score of each test instance according to (1) and (2) (line 7). Afterward, we check the position of the current vector $\vec{\mathbf{v}}$ in the feature space and get the verdict of the monitor accordingly. Finally, we store the instance index along with Gini or entropy score to the corresponding verdict dictionary D (lines 9-14).

2) Prioritization algorithm: We rank the unlabeled test instances according to the monitor verdict into three ordered groups: rejected, uncertain (suspicious), and accepted (line 17). Following this order, we expect that most of the misclassified instances are mainly in the rejected group. A few misclassified

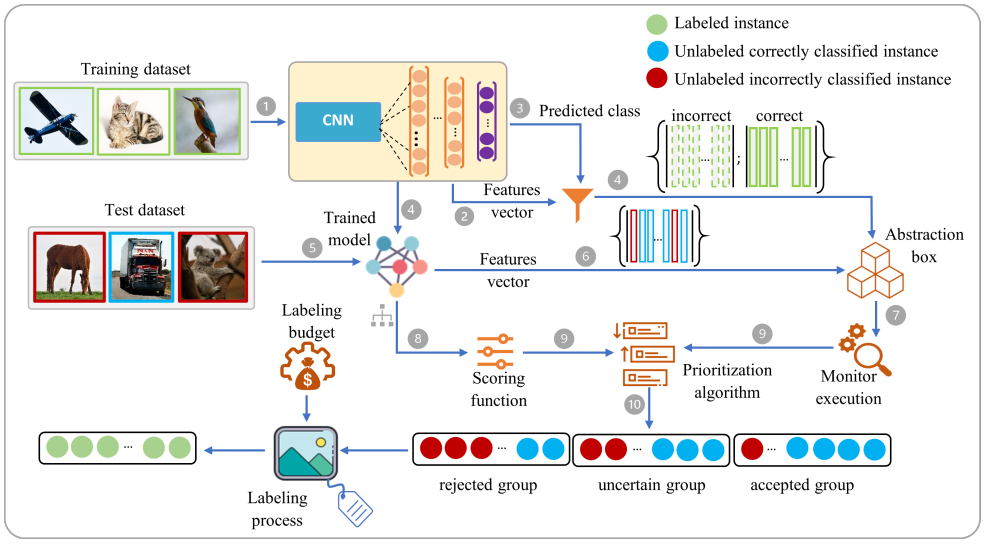


Fig. 3: The workflow of DeepAbstraction illustrates the main components and stages.

instances are in the uncertain group. A minimal number of misclassified instances are in the accepted group. Each group has two main types of instances, either correctly or incorrectly classified instances, as summarized in Table I. The instances in each group are in random order. Furthermore, the main goal of test prioritization is to prioritize the misclassified instances or the error-revealing instances: the true positives, uncertain positives, and false negatives in the rejected, uncertain, and accepted groups, respectively. Since the network easily and correctly classifies most instances, the Gini index and entropy score many instances with zero value. The zero-scored instances are often false positives, uncertain negatives, and true negatives. Thus, each group should not contain these instances to save the budget. In other words, we can save the budget by removing false positives (FPs) from the first group and labeling more uncertain positives (UPs) in the uncertain group. Similarly, removing the zero-score instances in the uncertain group allows for labeling more false negatives in the accepted group (line 19). We will analyze empirically the efficacy of this step over different benchmarks in Sec. V-A. Finally, we sort the instances in descending order based on the assigned score by the scoring function and select a certain number of instances according to the labeling budget (lines 20-23).

Algorithm 1: Test prioritization algorithm

Input: $\mathrm{X}_{\text{test}} = \{x_1, x_2, ..., x_n\}, \mathrm{B}^{\mathrm{c}}, \mathrm{B}^{\mathrm{inc}}$
$\mathcal{N}$: DNN, $bdgt$: *labeling budget*
Output: $\mathrm{X}_{\text{prioritized}} : \{x_1, ..., x_{bdgt}\}$
/* Monitors execution */
1 $\mathrm{D} \leftarrow \{\}$
2 **foreach** $x \in \mathrm{X}_{\text{test}}$ **do**
3 $idx \leftarrow \textbf{index}(x)$
4 $\vec{\mathbf{v}} \leftarrow \textbf{extract}(x)$
5 $y' \leftarrow \textbf{classify}(\mathcal{N}, x)$
6 $output \leftarrow \textbf{softmax}(\vec{\mathbf{v}})$
7 $score \leftarrow \textbf{gini\_index}(output)$ // or entropy
8
9 **if** $(\vec{\mathbf{v}} \in \mathrm{B}^{\mathrm{c}}[y']) \wedge (\vec{\mathbf{v}} \in \mathrm{B}^{\mathrm{inc}}[y'])$ **then**
10 $\mathrm{D}[uncertain].append([idx, score])$
11 **else if** $\vec{\mathbf{v}} \in \mathrm{B}^{\mathrm{c}}[y']$ **then**
12 $\mathrm{D}[accepted].append([idx, score])$
13 **else**
14 $\mathrm{D}[rejected].append([idx, score])$
/* Prioritization algorithm */
15 $sorted\_indx \leftarrow [\,]$
16 $prioritized\_indx \leftarrow [\,]$
17 $verdicts \leftarrow [rejected, uncertain, accepted]$
18 **foreach** $i \in verdicts$ **do**
19 $\mathrm{D}[i] \leftarrow \textbf{remove\_zero}(\mathcal{D}[i])$
20 $indices\_lst \leftarrow \textbf{sort}(\mathcal{D}[i])$
21 $sorted\_indx.extend(indxs\_lst)$
22 $prioritized\_indx \leftarrow \textbf{prioritize}(sorted\_indx, bdgt)$
23 $\mathrm{X}_{\text{prioritized}} \leftarrow \mathrm{X}_{\text{test}}[prioritized\_indx]$
24 **return** $\mathrm{X}_{\text{prioritized}}$

IV. Experimental Setup

In Sec. IV-A, we introduce the datasets and models by which we conduct our experiments. In Sec. IV-C and Sec. IV-B, we present the evaluation metric and the state-of-the-art baselines used to compare our framework. At last, we outline research questions in Sec. IV-D.

A. Datasets and DNN Models

We conduct our experiments on the most widely used datasets in image classification problems: MNIST [14], Fashion-MNIST [15], CIFAR10 [16], and SVHN [17].

TABLE I: Verdict types and definitions over all groups.

Verdict Type	Definition
True Positive (TP)	monitors truly reject the incorrect prediction
False Positive (FP)	monitors falsely reject the correct prediction
Uncertain Positive (UP)	monitors are uncertain towards incorrect prediction
Uncertain Negative (UN)	monitors are uncertain towards correct prediction
True Negative (TN)	monitors truly accept the correct prediction
False Negative (FP)	monitors falsely accept the incorrect prediction

We select pretrained CNN-based models that achieve state-of-the-art performance. Different DNN architectures (LeNet-5 [18], ResNet18 [19], VGG16 [20], GoogleNet [21], and AlexNet [22]) demonstrate that our framework generalizes well. Table II summarizes the details of applied datasets and networks. Further experiments are in the online repository. [1]

B. Baselines

We compare our framework to the following recent work in test prioritization, detailed in Sec. VI. We exclude the random approach since it achieves poor results [5]:

- Entropy [2] is the earliest work in test prioritization. It measures how much the neural network is uncertain about the test input.
- Distance-based Surprise Adequacy (DSA) [2] is a measure of how far the test instance is from the training data.
- DeepGini [3] is the technique that measures the likelihood of misclassification for a test input using the probability distributions of DNN output.
- TestRank [5] is the state-of-the-art technique in test prioritization. It leverages both intrinsic attributes from the penultimate layer and also contextual attributes to prioritize error-revealing instances.

C. Evaluation metrics

In this section, we introduce two metrics to evaluate our framework. The first is the Average Test Percentage of Fault (ATPF) [5], which evaluates the effectiveness of test prioritization approaches. The second is a new metric, called distance ratio, by which we estimate the regions where DeepGini and DeepAbstraction are more effective in the feature space.

We follow the evaluation metric (ATPF) in TestRank [5] since ATPF considers the labeling budget. On the other hand, the approaches in [2], [3] evaluate the performance by Average Percentage of Fault Detected (APFD) [23] mainly used in software test prioritization. Since there is no labeling cost in software test prioritization, APFD does not consider the labeling cost. Therefore, it is inapplicable to use APFD in test prioritization for deep learning networks.

ATPF is a measure of effectiveness to evaluate test prioritization techniques within a limited budget. We apply ATPF when the labeling budget (i) is less than or equal to the total number of error-exposing instances (N) in the test dataset.

TABLE II: Datasets and DNN architectures with the corresponding training and test accuracies.

Exp. ID	Dataset	Training dataset	Test dataset	Pr-trained DNN model	Training Acc. (%)	Test Acc. (%)
A	MNIST	60,000	10,000	LeNet5	99.67	98.54
B	MNIST	60,000	10,000	ResNet18	99.68	99.43
C	F-MNIST	60,000	10,000	VGG16	92.04	91.35
D	F-MNIST	60000	10000	ResNet18	94.97	92.47
E	CIFAR-10	50,000	10,000	GoogleNet	94.70	91.03
F	CIFAR-10	50,000	10,000	ResNet18	88.81	85.43
G	SVHN	73,257	26,032	AlexNet	94.63	93.23
H	SVHN	73,257	26,032	ResNet18	96.92	95.52

[1] https://github.com/verimag/DeepAbstraction

In other words, ATPF is the summation of the ratio between the number of error-revealing instances detected within the labeling budget (i) over the labeling budget (i):

$$\text{ATPF} = \frac{1}{N}\sum_{i=1}^{N}\frac{\text{number of prioritized instances}}{i} \tag{5}$$

We need to define a distance-based metric that can help compute whether the misclassified instance location is near the centroid or the classification boundary. Thus, let a set of centroids $\text{C} = \{\vec{\mathbf{c}}_1, \vec{\mathbf{c}}_2, ..., \vec{\mathbf{c}}_g\}$ where g is the total number of classes in the dataset. Also, u is the number of instances that belong to a certain class, and c denotes the centroid vector of a certain class, computed as follows:

$$\vec{\mathbf{c}} = \frac{\sum_{i=1}^{u}\vec{\mathbf{v}}_i}{u} \tag{6}$$

We define **a distance ratio** as a ratio of the Euclidean distance between an instance and the centroid of the predicted class y' to the Euclidean distance between an instance and the centroid of the ground truth y, defined formally as follows:

$$d(x) = \frac{\|\vec{\mathbf{c}}_{y'} - \vec{\mathbf{v}}\|_2}{\|\vec{\mathbf{c}}_y - \vec{\mathbf{v}}\|_2} \tag{7}$$

Based on the distance ratio, we group misclassified instances into three main regions:

1) If $d(x) \in\ ]0.0, 0.7]$, a misclassified instance becomes a **near-centroid** instance.
2) If $d(x) \in [0.7, 1.3]$, a misclassified instance is considered to be a **near-boundary** instance.
3) If $d(x) \in\ ]1.3, \infty]$, a misclassified instance becomes also a **near-centroid** instance.

The difference between the first and third regions is that, in region 1, an instance is close to the centroid of the incorrectly predicted class. In contrast, in region 3, an instance is marginally close to the centroid of the ground-truth class where the neural network still misclassifies this instance.

Since defining an accurate decision-boundary region in feature space is often complicated, we need to estimate the near-boundary region. We found through extensive experimental studies that this region is approximately defined by a distance ratio of 1.0 ± 0.3. Note that when the distance ratio is 1, instances are more likely to be very close to the decision boundary.

D. Research Questions

We empirically evaluate DeepAbstraction from three perspectives: effectiveness, efficiency, and performance stability. We answer the following questions:

RQ1 (effectiveness): Is DeepAbstraction more effective than other DL test prioritization methods in prioritizing more error-revealing instances?

RQ2 (efficiency): What is the time complexity of DeepAbstraction to prioritize the unlabeled test dataset?

RQ3 (stability): How does the clustering parameter τ affect the effectiveness of DeepAbstraction?

V. Experimental Evaluation

This section aims at answering the research questions raised in Sec. IV-D. We conducted our experiments on a laptop with Intel(R) Core(TM)i7-7600U CPU, 2.80 GHz, and 8GB RAM. We implemented our tool on top of the open-source framework PyTorch v1.9.0. The source code can be accessed here. [2]

A. RQ1: Effectiveness

To answer **RQ1**, we make the following comparisons: first, we compare DeepAbstraction with DSA, Entropy, and DeepGini. Then, due to the particular data split of TestRank, we compare DeepAbstraction separately to TestRank (we follow the same dataset splits).

Table III overviews the effectiveness comparison between DeepAbstraction and the other baselines. Overall, DeepAbstraction outperforms other approaches significantly in all datasets. DeepAbstraction is much more effective than DSA, e.g., experiment (D, FMNIST) in Table III shows an ATPF improvement of 39.15%. Moreover, our framework achieves better results than Entropy and DeepGini, with a considerable margin ranging between 11.54% $\sim$ 30.11% in terms of ATPF.

As shown in Table IV, our framework remarkably outperforms TestRank in 4 out of 6 experiments. Additionally, another advantage of our framework over TestRank is that our framework avoids the pre-labeling efforts in TestRank. For instance, TestRank in experiments (A, CIFAR10) and (C, SVHN) pre-labeled 8,000 and 10,000 test instances to operate, respectively, which is very time-consuming. The comparison in Table IV is unfair since TestRank requires more budget, i.e., pre-labeling cost. If we count this pre-labeling cost into the budget of TestRank, we find out that DeepAbstraction is considerably superior to TestRank in all datasets.

Next, we investigate why our framework is more effective than other techniques in two aspects: i) the regions where other methods fail to prioritize error-revealing instances; ii) the removal of zero-scored instances inside each group, as mentioned in line 19 of algorithm 1.

Regarding the regions, Table V shows that there is a significant number of error-revealing instances residing close to the centroids in column *total*. DeepGini and Entropy can poorly estimate the error-revealing capability of test instances residing in regions 1 and 3 (near-centroid regions), as defined in Sec. IV-C. On the contrary, DeepAbstraction works effectively in all regions, as demonstrated in almost all experiments in Table V. For instance, the results of experiment E show that DeepGini and DeepAbstraction prioritize approximately the same number of near-boundaries instances (371 versus 373). However, DeepAbstraction prioritizes 202 near-centroid instances highly greater than 65 instances prioritized by DeepGini.

As for the removal of zero-scored instances, Fig. 4 shows that this step reduces considerably the number of false positives (defined in Table I), particularly in FMNIST and MNIST datasets. Although, there is a slight drop in the number of true positives. This drop is due to the GI function weakness that scores some true positives with zero scores, e.g., these instances in regions 1 and 3. In a nutshell, the removal process for zero-scored instances improves the effectiveness of DeepAbstraction by prioritizing more uncertain positives from the second group.

TABLE III: Effectiveness comparison between DeeAbstraction and other baselines in terms of ATPF(%).

Exp. ID	Dataset	DSA	Entropy	DeepGini	DeepAbstr.	Δ
A	MNIST	33.15	34.67	50.36	**61.90**	↑+11.54
B	MNIST	27.84	29.44	33.98	**57.58**	↑+23.60
C	F-MNIST	41.77	42.59	56.23	**86.34**	↑+30.11
D	F-MNIST	43.66	41.21	55.27	**82.81**	↑+27.54
E	CIFAR-10	43.13	42.72	57.17	**79.12**	↑+21.95
F	CIFAR-10	47.98	52.73	60.20	**83.10**	↑+22.90
G	SVHN	47.44	50.73	59.92	**83.60**	↑+23.68
H	SVHN	47.22	44.54	55.06	**83.82**	↑+28.76

TABLE IV: Effectiveness comparison between TestRank and DeepAbstration in terms of ATPF(%).

Exp. ID	Model	Dataset	Validation Acc. (%)	TestRank (%)	DeepAbstr. (%)	Δ
A	ResNet-18	CIFAR 10	70.10	87.87	84.15	↓- 03.72
B	ResNet-18	CIFAR 10	66.40	85.53	89.31	↑+03.78
C	ResNet-18	CIFAR 10	68.30	76.56	85.02	↑+08.46
D	Wide-ResNet	SVHN	94.20	76.36	85.91	↑+09.55
E	Wide-ResNet	SVHN	92.50	66.06	83.19	↑+17.13
F	Wide-ResNet	SVHN	81.60	95.32	86.89	↓- 08.43
Total						↑+26.77

TABLE V: The number of instances DeepAbstraction and DeepGini detect in each region

Exp. ID	Dataset	Near-Boundary instances			Near-Centroid instances		
		total	DeepAbst.	DeepGini	total	DeepAbst.	DeepGini
A	MNIST	92	**56**	49	54	**40**	13
B	MNIST	31	**28**	14	26	**16**	6
C	F-MNIST	450	**303**	229	415	**306**	191
D	F-MNIST	478	**312**	301	275	**212**	72
E	CIFAR-10	630	**373**	371	267	**202**	65
F	CIFAR-10	1017	**639**	631	440	**309**	113
G	SVHN	1319	**842**	736	444	**269**	171
H	SVHN	760	464	**501**	405	**344**	50

RQ1 Answer :

DeepAbstraction is more effective than other test prioritization techniques, including state-of-the-art technique (TestRank). Contrary to the existing work, DeepAbstraction can effectively estimate the error-revealing capability of test instances in all regions.

[2] https://github.com/verimag/DeepAbstraction

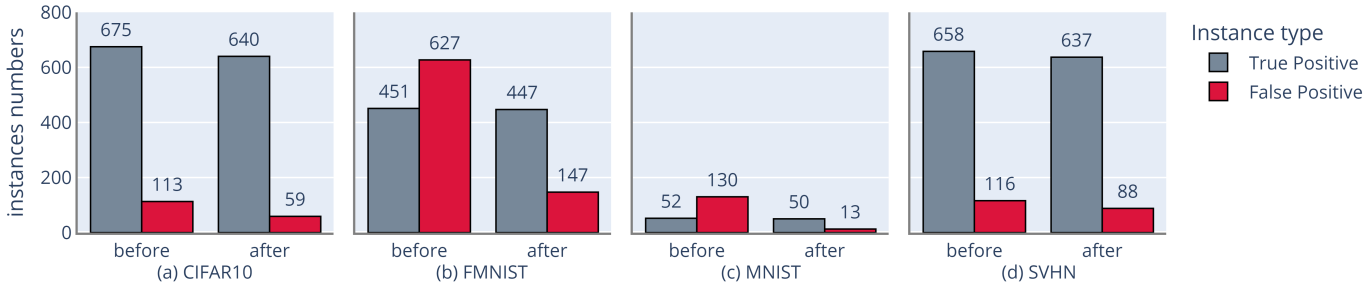


Fig. 4: The number of true positives and false positives before and after removing zero-scored instances.

B. RQ2: Efficiency

DeepAbstraction consists of two main components: monitors and prioritization algorithm. According to [10], the complexity of monitors construction is the complexity of the clustering $\mathcal{O}(m^2)$ where m is a number of training samples. The complexity of the membership query and scoring function for test instances is $\mathcal{O}(n)$ where n is the number of test instances. While the complexity of sorting algorithm is $\mathcal{O}(n \times \log n)$. Therefore, the overall complexity of DeepAbstraction is $\mathcal{O}(m^2)$. Although, the complexity can be approximately linear $\mathcal{O}(m)$, if we follow k-means algorithm with cluster shifting [24]. In addition, the time used for clustering is very small compared to the labeling time for the entire dataset.

We compare the time complexity between both DeepAbstraction and TestRank, and find that DeepAbstraction is much more efficient than TestRank with a large margin for three main reasons. First, TestRank requires pre-labeling for some unlabeled test instances as a prerequisite. Second, TestRank utilizes kNN graph for similarity measure which complexity is $\mathcal{O}((m+n)^2)$. Third, TestRank consists of two neural networks: Graph Neural Network and MLP Multi-Layer Perceptron. Training and evaluation for these two neural networks increases the time complexity of TestRank exhaustively.

RQ2 Answer :

DeepAbstraction is an efficient framework and the time complexity is $\mathcal{O}(m^2)$ which can be improved to be approximately linear with k-means algorithm with cluster shifting. Thus, the time complexity of DeepAbstraction is negligible compared to the time of manual labeling. In practice, DeepAbsraction is much more efficient than TestRank.

C. RQ3: Stability

In this section, we discuss the performance stability of DeepAbstraction in two aspects. First, we investigate how the hyperparameter τ affects our framework performance among different benchmarks. Second, we compare the performance stability of our framework to TestRank.

We recall that the smaller the τ is, the more compact the box abstraction is, and vice versa. Due to a limited number of misclassified training instances with very high accurate models, the box abstraction is too compact, which causes many false positives. For instance, in our experiments with validation accuracy greater than 98%, we observed many false positives during the test stage. We can solve this issue by selecting a larger value of τ to enlarge the box abstraction with more true negatives than false positives, e.g., the best τ value providing a stable ATPF in experiments A and B in Fig. 5 is 0.4. In the case of relatively high accurate models, we have coarse boxes, implying more false negatives in the testing phase. To this end, we select a smaller value of τ to have a more compact box abstraction with more true positives than false negatives. For instance, we find τ = 0.05 in experiments C to F provides more stable results in terms of ATPF(%).

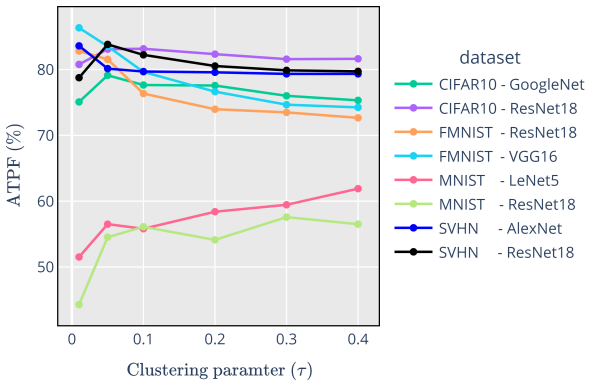


Fig. 5: The effect of clustering parameter τ on the performance stability of DeepAbstraction.

In comparison to TestRank, the results in column 6 of Table IV confirm the performance stability of DeepAbstraction among different datasets. Whereas column 5 shows that TestRank has high unstable performance, e.g., although experiments E and F are conducted on the same dataset and model, the ATPF results differ significantly, 95.32% vs. 66.06%.

RQ3 Answer :

The default value of τ is 0.05 to provide stable performance in DeepAbstraction. With very high accurate models, τ should be 0.4 to provide a stable performance. Furthermore, DeepAbstraction is more stable than the recent work (TestRank) on various benchmarks.

VI. Related Work

Byun et al. [2] first study the test prioritization problem for deep learning systems. The authors propose three sentiment measures: i) *Confidence* measured by softmax output layers; ii) *Surprise* measured by the Euclidean distance between the activation outputs of training data; iii) *Uncertainty* is measured by the probability distribution of the DNN's prediction. DeepGini [3] use only the Gini Index to estimate the error-revealing capability of each test instance. The main drawback of these two approaches is that they entirely depend on the model output to measure the uncertainty of the model, which can be unreliable [6].

PRIMA [4] is a mutation-based test prioritization technique. The critical issue is that random mutation in deep learning systems is still not safe, i.e., the mutated model may be a different model, and the results based on the mutated models might be misleading. One can use nonrandom mutation operators to improve this work. Furthermore, PRIMA requires storing multiple mutated models and mutated test inputs, which results in higher consumption of storage resources than other test prioritization techniques. Hence, PRIMA may be inapplicable to large state-of-the-art neural networks and large datasets.

TestRank [5] is the state-of-the-art test prioritization technique, which fixes the issue with the previous approaches [2], [3] by combining intrinsic attributes and the contextual attributes of the networks to measure the network's bug-revealing capability over test instances effectively. However, TestRank still suffers three limitations: 1) it requires prelabeling for some test instances to enable prioritizing other test instances; 2) TestRank is not applicable if the budget or the test dataset is too small since it requires a lot of labeled test data to operate; 3) It is also not stable, e.g., TestRank only works well with low accurate models, as shown in Table IV. On the other hand, our framework can operate without prelabeling and work with a small budget and small test dataset. Lastly, it shows stable performance.

VII. Conclusion and Future Work

We develop a 2-level prioritization framework, called DeepAbstraction, to estimate the error-reveling capability of unlabeled test instances. DeepAbstraction is composed of two ranking systems: monitors and scoring function. The use of monitors in DeepAbstraction helps detect more error-exposing instances in both regions: near the centroids and the decision boundaries. Experimental results demonstrate that DeepAbstraction is more effective and stable than other DL test prioritization methods, including state-of-the-art technique(TestRank). In our framework, we mainly investigated the capability of monitors in test prioritization. Hence, future work should concentrate more on other scoring functions. Moreover, monitors in our framework apply only to classification models. Therefore, we should carry out future work on using monitors for regression models. Lastly, future work should also study if the high-feature vector's length impacts the performance of monitors.

References

[1] G. Wang, G. Wang, X. Zhang, J. Lai, and L. Lin, "Weakly supervised person re-identification: Cost-effective learning with A new benchmark," *CoRR*, vol. abs/1904.03845, 2019.

[2] T. Byun, V. Sharma, A. Vijayakumar, S. Rayadurgam, and D. D. Cofer, "Input prioritization for testing neural networks," *CoRR*, vol. abs/1901.03768, 2019.

[3] Y. Feng, Q. Shi, X. Gao, J. Wan, C. Fang, and Z. Chen, "Deepgini: prioritizing massive tests to enhance the robustness of deep neural networks," *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2020.

[4] Z. Wang, H. You, J. Chen, Y. Zhang, X. Dong, and W. Zhang, "Prioritizing test inputs for deep neural networks via mutation analysis," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pp. 397–409, IEEE, 2021.

[5] Y. Li, M. Li, Q. Lai, Y. Liu, and Q. Xu, "Testrank: Bringing order into unlabeled test instances for deep learning tasks," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[6] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," *CoRR*, vol. abs/1506.02142, 2015.

[7] A. Subramanya, S. Srinivas, and R. V. Babu, "Confidence estimation in deep neural networks via density modelling," *CoRR*, vol. abs/1707.07013, 2017.

[8] S. Mani, A. Sankaran, S. Tamilselvam, and A. Sethi, "Coverage testing of deep learning models using dataset characterization," *arXiv preprint arXiv:1911.07309*, 2019.

[9] T. A. Henzinger, A. Lukina, and C. Schilling, "Outside the box: Abstraction-based monitoring of neural networks," *CoRR*, vol. abs/1911.09032, 2019.

[10] C. Wu, Y. Falcone, and S. Bensalem, "Customizable reference runtime monitoring of neural networks using resolution boxes," *CoRR*, vol. abs/2104.14435, 2021.

[11] L. Plata-Pérez, J. Sanchez-Perez, and F. Sánchez-Sánchez, "An elementary characterization of the gini index," *Mathematical Social Sciences*, vol. 74, pp. 79–83, 2015.

[12] L. Ceriani and P. Verme, "The origins of the gini index: extracts from variabilità e mutabilità (1912) by corrado gini," *The Journal of Economic Inequality*, vol. 10, no. 3, pp. 421–443, 2012.

[13] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[14] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[15] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *CoRR*, vol. abs/1708.07747, 2017.

[16] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," *University of Toronto*, 2009.

[17] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," *NIPS*, 2011.

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

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

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

[21] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014.

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[23] R. Pradeepa and K. VimalDevi, "Effectiveness of testcase prioritization using apfd metric: Survey," in *International Conference on Research Trends in Computer Technologies (ICRTCT—2013). Proceedings published in International Journal of Computer Applications®(IJCA)*, pp. 0975–8887, 2013.

[24] M. K. Pakhira, "A linear time-complexity k-means algorithm using cluster shifting," in *2014 international conference on computational intelligence and communication networks*, pp. 1047–1051, IEEE, 2014.