DeepRank: Test Case Prioritization for Deep Neural Networks

Wei Li[1], Zhiyi Zhang[1,2,*], Yifan Jian[3], Chen Liu[4,*] and Zhiqiu Huang[1,5]
[1]Collage of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China
[2]Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing, China
[3]Science and Technology on Reactor System Design Technology Laboratory, Nuclear Power Institute of China, Chengdu, China
[4]3School of Marxism, Yangzhou University, Yangzhou, China
[5]Ministry Key Laboratory for Safety-Critical Software Development and Verification,
Nanjing University of Aeronautics and Astronautics, Nanjing, China
wei_sz2116@nuaa.edu.cn, zyzhang10@nuaa.edu.cn, ncepujyf@163.com, lauchan@yzu.edu.cn, zqhuang@nuaa.edu.cn

***Abstract*—Deep neural networks (DNNs) have been widely used in safety-critical fields such as autonomous driving and medical diagnosis. However, DNNs are easily disturbed to make wrong decisions, which may lead to loss of life or property. Therefore, it is vital to test DNN adequately. In practice, to reveal the incorrect behavior of DNN and improve its robustness, testers usually need massive labeled data to test and optimize DNN. However, labeling test inputs to detect the correctness of DNN predictions is an expensive and time-consuming task that even affects the efficiency of DNN testing.**

To relieve the labeling-cost problem, we propose DeepRank, a test case prioritization technique based on cross-entropy loss. The key idea of DeepRank is that the higher the loss value of a test case relative to the DNN, the more likely it is to be mispredicted and the more conducive it is to improve the robustness of the DNN through retraining. Therefore, the cross-entropy loss value can be used for test case prioritization. We experimentally validate our approach on two datasets and three DNNs models. The experimental results demonstrate that DeepRank is significantly better than existing test case prioritization methods regarding fault-revealing capability and retraining effectiveness.

***Index Terms*—DNN Testing, Cross Entropy, Test Case Prioritization**

I. Introduction

Deep neural networks (DNNs) have made breakthroughs in many fields, such as image recognition, speech recognition, and natural language processing. They have been widely integrated into software systems to help solve various tasks, such as autonomous driving systems, medical diagnostic systems, etc. However, DNNs are susceptible to interference to make bad decisions that can lead to loss of life or property, such as the fatal crash of Google's self-driving car [18]. Therefore, it is vital to ensure the reliability and robustness of software systems driven by DNN.

DNN testing is one of the most effective ways to guarantee its quality [14], [17]. However, unlike traditional software testing, DNN is based on a data-driven programming paradigm that uses massive data to be trained to form internal logic [7], which makes DNN models have the characteristics of poor interpretability and low generalization ability. As a result, many traditional software testing methods cannot be directly applied to DNN testing. To adequately test DNN models, testers often require massive labeled data to test and optimize DNN models. However, labeling test cases to verify the correctness of DNN is costly. There are three main reasons: first, the scale of test cases that need to be labeled is large; Second, it mainly relies on manual labeling, and the labeling efficiency is low; Finally, test case labeling usually requires professional knowledge in specific fields [15].

To relieve the labeling-cost problem, a feasible solution is to prioritize unlabeled test cases and give higher priority to test cases that could lead DNN to make wrong decisions. In this paper, we propose DeepRank, a test case prioritization technique based on cross-entropy loss. The key idea of DeepRank is that the higher the loss value of a test case relative to the DNN, the more likely it is to be mispredicted and the more conducive it is to improve the robustness of the DNN through retraining. Therefore, the cross-entropy loss value can be used for test case prioritization. We only label test cases with high priority after prioritization, which can save labeling costs and improve the efficiency of DNN testing. We designed four sets of experiments for empirical research on two commonly used datasets in image recognition and three DNNs with different structures. The experimental results demonstrate that DeepRank is significantly better than existing test case prioritization methods regarding fault-revealing capability and retraining effectiveness.

The main contributions of this paper are as follows:

- We propose DeepRank, a test case prioritization technique, which can reduce labeling costs and improve the efficiency of DNN testing.
- We use cross-entropy loss value for test case prioritization and prove the effectiveness of our method through experiments.
- The test cases with high priority after prioritization by our method can be used to guide DNN retraining and improve DNN robustness.

*corresponding author
DOI reference number: 10.18293/SEKE2023-188

The rest of the paper is organized as follows. Section II introduces background on DNN and test case prioritization. Section III describes the implementation of our method. Section IV details the setup of the experiment. Section V analyzes the experimental results and demonstrates the effectiveness of our method. Section VI concludes this paper.

II. BACKGROUND

This section includes basic knowledge about deep neural networks (DNNs), neuron coverage metrics, and test case prioritization methods for DNNs.

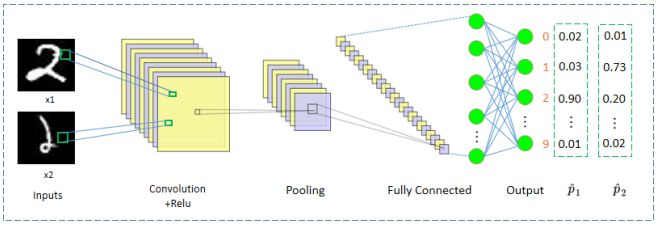

Fig. 1: An example to illustrate the CNN structure

A. Convolutional Neural Network

Convolutional neural networks (CNNs) are the core of image processing tasks. As shown in Fig. 1, a CNN consists of multiple layers, i.e., convolution layer, pooling layer, and fully connected layer. The convolution layer is used to extract the features of the input data. The pooling layer is periodically inserted between successive convolutional layers to reduce the number of parameters in CNN and effectively prevent overfitting. The fully connected layer is used to map the learned feature representation to the label space of the input.

Generally speaking, CNN maps the input data x to the output result y. For example, in an N classification task, given an input, after processing by CNN internal neurons, an N-dimensional vector $out = \{v_1, v_2, ..., v_N\}$ will be obtained in the output layer and then normalized using the softmax function [2], a set of probability vectors $\hat{p}_i = \{\hat{p}_{i,1}, \hat{p}_{i,2}, ..., \hat{p}_{i,N}\}$ will be obtained, $\hat{p}_{i,j}$ represents the probability that the neural network predicts the test case x_i as an j-th class, and the final prediction result of CNN is the category corresponding to the value with the highest probability in $\hat{p}_i$.

B. Neuron Coverage Metric

Inspired by code coverage in traditional software testing, researchers have recently combined coverage with neurons to propose a series of neuron coverage metrics for DNN testing. This section briefly describes the DNN test method guided by the neuron coverage under investigation.

Neuron Activation Coverage (NAC(k)) [12]. The metric defines neuron coverage in DNN as: if the output value of a neuron is greater than the threshold k, the neuron is considered to be covered. The fundamental hypothesis of NAC(k) is that the greater the number of neurons covered, the more DNN states are explored. For a test case, NAC(k) is calculated as the ratio of the number of neurons covered by that test case to the total number of neurons in the DNN.

Neuron Boundary Coverage (NBC(k)) [9]. SBC(k) first counts the upper boundary $high_n$ and lower boundary low_n of the output value of each neuron in the DNN on the training set. They refer to $(-\infty, low_n) \cup (high_n, +\infty)$ as the corner-case regions of a neuron n. This method focuses on measuring test cases coverage in corner-case regions. Since each neuron has one upper bound and one low bound, for a test case, SBC(k) is calculated as a ratio of the number of neurons covered by a corner-case region to twice the total number of neurons in the DNN.

Strong Neuron Activation Coverage (SNAC(k)) [9]. It can be seen as a special case of NBC(k) as it only considers coverage of the upper boundary region $(high_n, +\infty)$. For a test case, SNA(k) is calculated as the ratio of the number of neurons covered by the upper boundary to the total number of neurons in the DNN.

Top-k Neuron Coverage (TKNC(k)) [9]. TKNC(k) focuses on the k neurons that are the most active in each layer of the DNN. It is defined as the ratio of the total number of top-k neurons on each layer to the total number of neurons in the DNN.

C. Test Case Prioritization

Test case prioritization refers to rearranging the execution order of test cases in a test set according to predetermined criteria so that high-priority test cases are executed earlier in the test execution process than low-priority test cases. Two main coverage-based test case prioritization techniques are known as the Coverage-Total Method (CTM) and the Coverage-Additional Method (CAM) [16].

Coverage-Total Method (CTM). The coverage of each test case is calculated first, and then the individual test cases are prioritized based on their total coverage. When multiple test cases have the same coverage, the relative order of these test cases is randomly determined. Assuming that there are n test cases in the test set T and m coverage entities in program P, the time cost of CTM is $O(mn)$.

Coverage-Additional Method (CAM). The idea of CAM is that if a test case can cover as many entities as possible that were not covered by previously executed test cases, the higher the priority of that test case. Because such a test case is most likely to expose errors not exposed by the previously executed test case. Assuming that there are n test cases in the test set T and m coverage entities in program P, the time cost of CAM is $O(mn^2)$.

III. OUR APPROACH

We propose a test case prioritization method DeepRank, based on the cross-entropy loss value of test cases. First, we introduce the motivation of DeepRank. Then, we introduce the overall framework of DeepRank. After that, we introduce the specific implementation steps of DeepRank. Finally, we introduce how to use DeepRank to guide the retraining of DNN models to improve their robustness.

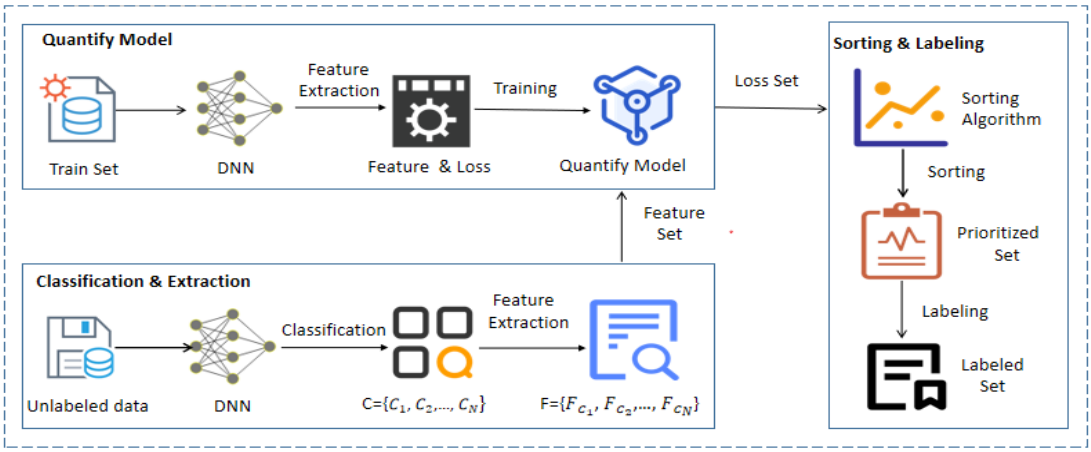

Fig. 2: Overview of DeepRank

A. Motivation

Unlike the DNN test case prioritization method based on uncertainty [3] and neuron coverage [6], [9], [12], DeepRank takes the cross-entropy loss value of a test case relative to the DNN as the prioritize metric, and the greater the loss, the greater the probability of DNN misprediction.

For a trained DNN, the higher the loss value of a test case relative to the DNN, the more likely it is to be mispredicted. For example, as shown in Fig.1, we select two test cases x_1 and x_2 with actual labels 2 from the MNIST dataset, input them into the trained LeNet1 model and get the prediction results $\hat{p}_1 = \{0.02, 0.03, 0.90, ..., 0.01\}$, $\hat{p}_2 = \{0.01, 0.73, 0.20, ..., 0.02\}$, respectively. Therefore, LeNet1 predicts x_1 as 2 but x_2 as 1. It can be seen from the prediction results that the x_1 prediction is correct, but the x_2 prediction is wrong.

$$CEloss = -\sum_{n=1}^{N} p_{i,n} \log \hat{p}_{i,n} \tag{1}$$

According to the cross entropy loss function (CEloss) in (1), where N is the number of output classes. It can be calculated that the loss values of x_1 and x_2 relative to the LeNet1 model are $loss_1$=0.046 and $loss_2$=0.699, respectively.

From this example, it can be found that the cross-entropy loss value of x_2 relative to the DNN is greater than x_1, and the probability that the DNN incorrectly predicts x_2 is greater than x_1. The greater the loss value of a test case for the DNN, the greater the probability of the model misprediction. Therefore, the cross-entropy loss value can be used for test case prioritization. The greater the loss, the higher the priority.

B. Overview of DeepRank

Fig. 2 shows the overall framework of DeepRank. In general, the implementation of DeepRank can be divided into three parts. First, train a model for quantifying the cross-entropy loss value of test cases. Second, input the test cases into the DNN, divided into N sets according to the prediction class of DNN, and the features of the test cases in different sets are extracted in turn. Then the extracted features are input into the cross-entropy loss quantification model constructed in the first step. The cross-entropy loss value of the test case relative to the DNN can be obtained. Finally, prioritization according to the cross-entropy loss value of the test cases. Test cases with large cross-entropy loss values are given higher priority. Then selects high-priority test cases from each category collection to label.

C. Test Case Prioritization Process

- Step1. **Build the dataset**: We use the training set in the original dataset as the initial data because the cross-entropy loss value of each sample in the training set relative to the DNN can be easily obtained, and the training set and the test set are independent of each other. To increase the generalization ability of the cross-entropy loss quantification model, we add the adversarial samples generated by FGSM [4] and PGD [10] into the training set to form a new dataset $X = \{x_1, x_2, x_3, ..., x_m\}$.
- Step2. **Feature extraction**: Let $L = \{e_1, e_2, e_3, ..., e_n\}$ represent the set of neurons in the last hidden layer in the DNN, $\alpha(x)$ represents the output value of the neuron e relative to x, and $\alpha_L(x)$ represents the Activation Trace (AT) [6] of neurons in L, that is, the set of output values of all neurons in the L relative to x. $\alpha_L(X) = \{\alpha_L(x) | x \in X\}$ denote the AT of neurons in L on X. Then count the range of output values of each neuron e_i in L on X: $[low_i, high_i]$, and divide it into k equal intervals $\alpha_e(X) = \{u_1, u_2, u_3, ..., u_k\}$, in this paper, $k = 100$, if $\alpha_{e_i}(x) \in u_j$, let $f_x(e_i) = j$, therefore, for each x in X can extract an n-dimensional feature vector $F(x) = \{f_x(e_1), f_x(e_2), ..., f_x(e_n)\}$, where $f_x(e_i) \in [1, k]$.
- Step3. **Extract the label**: After input X to DNN, the output value of neurons in the output layer of DNN is processed by the softmax activation function to obtain the set of predicted probabilities $P(X) = \{p(x) | x \in X\}$, where $p(x) = \{p_1, p_2, ..., p_N\}$, N represents the number of categories. Then the cross-entropy loss value of x is

calculated according to the actual label as the label of the cross-entropy loss quantification model.

- Step4. **Train model**: Use the features extracted in step 2 and the labels calculated in step 3 as the training set of the model to quantify the test case cross-entropy loss value. Specifically, DeepRank uses XGBoost [1] to build a cross-entropy loss quantification model, which is one of the most popular algorithms in the field of machine learning with massively parallel computing power and sound portability and can effectively learn more complex features from basic features, so XGBoost is very suitable for solving our problem.
- Step5. **Process unlabeled data**: Input the unlabeled test set T into the DNN, divide T into N sets $C = \{C_1, C_2, ..., C_N\}$ according to the classification results of the DNN, and then use the method in step 2 to extract the features of C_i to obtain $Fc = \{F_{C_1}, F_{C_2}, ..., F_{C_N}\}$, and then input the features into the model trained in step3 to obtain the set of cross-entropy loss values $Loss = \{L_1, L_2, ..., L_N\}$ of each test case relative to the DNN.
- step6. **Sorting and labeling**:Use the quicksort algorithm to sort L_i in descending order to obtain the sorted unlabeled test set $R = \{R_1, R_2, ..., R_N\}$. Finally, To evenly select data from each category, according to the test budget, select the first n test cases from R_i (i=1, 2,...,N) to label.

There are two reasons why features are extracted from AT in L. First, the output of neurons in L is generally regarded as a learned representation of the training data. When the operating context changes, the representation is more stable than the prediction, and this is supported by transfer learning practices, where only the SoftMax layer is retrained for different tasks [5]. Second, DNN prediction comes directly from the linear combination of the output of this layer, so it must be highly correlated with the prediction accuracy [8].

D. Enhancing DNN with DeepRank

Since DNN is a data-driven programming paradigm, we cannot fix software bugs by directly modifying the code like traditional software development. Still, we can add as much data as possible to the DNN training set and retrain the DNN to enhance its robustness. However, in the actual scenario, a large amount of data collected is unlabeled and requires expensive labeling for DNN retraining. The main idea of DeepRank is that the greater the loss value of a test case relative to the DNN, the greater the probability of DNN misprediction and the more conducive it is to improve the robustness of the DNN through retraining. Therefore, the cross-entropy loss value can be used for test case prioritization. We only label test cases with high priority after prioritization, which reveals more DNN defects within a limited test budget.

In short, DeepRank can not only for test case prioritization but also use test cases with high priority after prioritization by DeepRank add into the training set to retrain DNN to improve the robustness of DNN.

IV. EXPERIMENTS

This section describes the experimental setup, including the datasets and DNN models used in the experiment, the construction candidate dataset, and the research questions.

A. Datasets and Models

TABLE I: DATASETS AND DNN MODELS

Dataset	DNN Model	Layers	Neurons	Train set	Test set
MNIST	LeNet-1	5	42	60000	10000
	ResNet-20	20	698	60000	10000
SVHN	VGG-16	21	7274	73257	26032
	ResNet-20	20	698	73257	26032

As shown in Table I, to evaluate our proposed method, we selected two widely used public datasets in the field of image recognition: MNIST and SVHN [11], and three DNN models with different structures and scales: LeNet-1, ResNet-20, and VGG-16 [13]. Among them, MNIST is a handwritten digit recognition dataset containing 7,000 grayscale images with a size of 28*28, of which 60,000 are training sets, and 10,000 are test sets, with a total of 10 categories. SVHN was collected from house numbers in Google Street View imagery and contained over 60,000 color images with a size of 32*32. To increase the reliability of experimental results, we selected two different DNN models for each dataset and designed four sets of experiments.

B. Construction Candidate Dataset

Although DNNs are carefully trained to predict high accuracy on the original test set, they are highly inaccurate for some corner test cases, so exploring the prioritization of these data is necessary. We use two commonly used adversarial sample generation methods, FGSM [4] and PGD [10], to generate these corner test cases for each dataset. We generate data the same size as the original test set for each adversarial sample generation method and evenly partition the original test set and the generated adversarial data into a new testing set T, and a new validation set V. For the MNIST dataset, we have new test and validation sets of size 15,000 each, where 5,000 are original test sets and the other 10,000 are adversarial samples generated by FGSM and PGD.

C. Research Questions

RQ1. Quantify: Can DeepRank accurately quantify the loss value of a test case relative to the DNN?

We use the R^2 Score and the Root-Mean-Square Error (RMSE) to answer RQ1. The R^2 Score is used to evaluate the regression model's fitting effect. The higher the coefficient of the R^2 Score, the closer it is to 1, and the better the fitting effect of the model. The RMSE is used to assess the accuracy of the regression model predictions.

RQ2. Effectiveness: Can DeepRank find a better permutation of tests than the baseline methods?

We collect the cumulative sum of the errors found by specific test case prioritization methods and calculate the corresponding RAUC (ratio of area under the curve) between the prioritization method and the theoretical curve.

RQ3. Enhancement: Can DeepRank guide the retraining of a DNN to improve its accuracy?

We evenly divide the original test set and the generated adversarial test cases into a test set T, and a validation set V, then prioritize T, and take the top 10% from the sorted T to the initial training set for retraining. Finally, we observe the accuracy of the DNN after retraining on the validation set V.

V. RESULT ANALYSIS

In this section, we present and analyze the results of our method on the quantification of test case loss values (RQ1) and the effectiveness of other baseline methods in test case prioritization (RQ2), and the improvement of accuracy after DNN retraining (RQ3).

TABLE II: R^2 SCORE AND RMSE

Dataset	DNN Model	R^2 Score	RMSE
MNIST	LeNet-1	94.39%	2.33
	ResNet-20	97.9%	2.86
SVHN	VGG-16	97.07%	3.49
	ResNet-20	98.68%	1.39

A. *RQ1. Quantify*

As shown in Table II, for each dataset and model combination, the R^2 Score of the test case cross-entropy quantization model trained by us is above 94%, especially for the SVHN and ResNet20 combination, the R^2 Score is 98.68%, which indicates that the model trained with the features we extracted has an excellent fit to the cross-entropy loss of the test case. In addition, we also provide the RMSE for each cross-entropy quantization model, and the results show that the prediction error of the quantization model remains within a low range.

To more intuitively show the effect of the test case cross-entropy loss quantification model, we randomly select 200 test cases from the test set T for each dataset and model combination. In Fig. 3, the x-axis represents the number of test cases and y-axis represents the cross-entropy loss value of the test case. The blue line represents the actual cross-entropy loss, and the orange line represents the cross-entropy loss values quantified by our method. The results show that the cross-entropy loss value quantified by our method is almost consistent with the actual value. It is worth noting that although the cross-entropy loss quantification of some test cases is not accurate enough, as long as it is the same as the actual cross-entropy loss value change trend. Because in practical applications, we only select a small number of test cases with large cross-entropy loss values from massive candidate sets for labeling to save labeling costs.

In summary, DeepRank can accurately quantify the cross-entropy loss value of test cases relative to DNNs within a small error range.

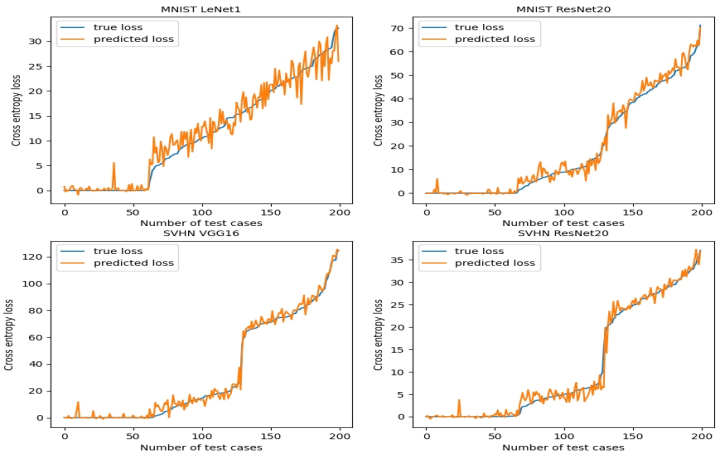

Fig. 3: Corss-entropy loss quantify effect

TABLE III: THE RAUC OF FAULT DETECTION

	Metrices	MNIST LeNet1	MNIST ResNet20	SVHN VGG16	SVHN ResNet20
Neuron Coverage	NAC(0.75)	0.583	0.854	0.603	0.994
	NBC(0)	0.702	0.975	0.769	0.995
	SNAC(0)	0.605	**0.999**	0.603	0.995
	TKNC(1)	0.75	0.747	0.753	0.75
Uncertainty	MaxP	0.809	0.845	0.763	0.755
	DeepGini	0.809	0.845	0.763	0.755
Our	DeepRank	**0.998**	0.993	**0.995**	**0.996**

B. *RQ2. Effectiveness*

We compare fault detection rates between DeepRank and other test case prioritization methods. For each dataset and DNN model combination, we calculate the RAUC on the test set T for each prioritization method. The closer the RAUC is to 1, the better the corresponding prioritization method works. As shown in Table III, DeepRank has achieved excellent results on all datasets and model combinations, i.e., RAUC is above 99%. In most cases, DeepRank is better than coverage-based and uncertainty-based methods. Taking the MNIST and LeNet1 combination as an example, DeepRank's RAUC is 0.998, and the NAC(0.75) is only 0.583. The uncertainty-based methods, such as MaxP and DeepGini, were close in all experiments. Overall, the uncertainty-based methods is superior to the coverage-based approachs but worse than DeepRank. Although NBC(0) and SNAC(0) were similar to DeepRank in the MNIST and ResNet20 combination and SVHN and ResNet20 combination, they were far less effective than DeepRank in the other two sets of experiments, indicating that the prioritization effect of NBC(0) and SNAC(0) was unstable. Conversely, the prioritization effect of Deeprank was stable in all experiments.

In summary, DeepRank achieves excellent prioritization effect in all combinations of datasets and models, and in most cases, DeepRank is better than coverage-based and

uncertainty-based methods.

TABLE IV: THE DNNS' ACCURACY VALUE AFTER RETRAINING WITH FIRST 10% PRIORITIZED TESTS.

	Metrices	MNIST LeNet1	MNIST ResNet20	SVHN VGG16	SVHN ResNet20
Neuron Coverage	NAC(0.75)	11.81	57.31	6.69	30.19
	NBC(0)	51.71	55.25	**36.89**	30.79
	SNAC(0)	25.83	55.39	12.39	30.72
	TKNC(1)	51.3	60.86	32.837	30.8
Uncertainty	MaxP	38.17	60.61	20.14	12.04
	DeepGini	39.3	60.68	14.13	11.78
Our	DeepRank	**51.9**	**61.22**	34.41	**31.22**

C. *RQ3. Enhancement*

For each dataset and model combination, we select the top 10% of the test cases prioritization by each test case prioritization method, add them to the initial training set for retraining the DNN, and then evaluate the effect of retraining by observing the improvement of the accuracy of the model on the validation set V. The results are shown in Table IV, and the accuracy of DNN models can be significantly improved by using DeepRank to guide retraining. For the MNIST and ResNet20 combination, DeepRank can improve the accuracy of DNNs on validation sets by 61.22%. In most cases, DeepRank is more effective at improving the model's accuracy than coverage-based and uncertainty-based methods. For example, combined with MNIST and LeNet1, DeepRank can improve model accuracy by 51.9%, but NAC(0) improves by 11.81% and MaxP by 38.17%. To some extent, this shows that the greater the loss value of the test case relative to the DNN, the more conducive it is to guide the retraining of the model.

In summary, DeepRank can effectively guide model retraining and improve model accuracy.

VI. CONCLUSION

In this paper, we propose Deeprank, a test case prioritization method based on cross-entropy loss. The key idea of DeepRank is that the higher the loss value of a test case relative to the DNN, the more likely it is to be mispredicted and the more conducive it is to improve the robustness of the DNN through retraining. Therefore, the cross-entropy loss value can be used for test case prioritization. We only label the test cases with higher priority, which can alleviate the cost of test case labeling and improve the efficiency of DNN testing. The experimental results show that DeepRank can effectively quantify the cross-entropy loss of test cases relative to DNN, has an excellent prioritization effect, and can guide DNN retraining, significantly improving the robustness of DNN.

ACKNOWLEDGMENT

This research is supported, in part, by National Natural Science Foundation of China (Grant No.62002162), and Natural Science Foundation of Jiangsu Province, China (Grant No.BK20200442), and Humanities and Social Sciences Fund of Yangzhou University (Grant No. xj2019-07), and Double-Innovation Doctor Program of Jiangsu Province, China (Grant No. (2019) 30755), and "Green Yang Jinfeng Project" Excellent Doctoral Program of Yangzhou, China (Grant No. (2019) 32).

REFERENCES

[1] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[2] John Denker and Yann LeCun. Transforming neural-net output levels to probability distributions. *Advances in neural information processing systems*, 3, 1990.

[3] Yang Feng, Qingkai Shi, Xinyu Gao, Jun Wan, Chunrong Fang, and Zhenyu Chen. Deepgini: prioritizing massive tests to enhance the robustness of deep neural networks. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 177–188, 2020.

[4] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[5] Jui-Ting Huang, Jinyu Li, Dong Yu, Li Deng, and Yifan Gong. Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 7304–7308. IEEE, 2013.

[6] Jinhan Kim, Robert Feldt, and Shin Yoo. Guiding deep learning system testing using surprise adequacy. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 1039–1049. IEEE, 2019.

[7] Hugo Larochelle, Yoshua Bengio, Jérôme Louradour, and Pascal Lamblin. Exploring strategies for training deep neural networks. *Journal of machine learning research*, 10(1), 2009.

[8] Zenan Li, Xiaoxing Ma, Chang Xu, Chun Cao, Jingwei Xu, and Jian Lü. Boosting operational dnn testing efficiency through conditioning. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 499–509, 2019.

[9] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang 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*, pages 120–131, 2018.

[10] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[11] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[12] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *proceedings of the 26th Symposium on Operating Systems Principles*, pages 1–18, 2017.

[13] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[14] Jialai Wang, Han Qiu, Yi Rong, Hengkai Ye, Qi Li, Zongpeng Li, and Chao Zhang. Bet: black-box efficient testing for convolutional neural networks. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 164–175, 2022.

[15] Zan Wang, Hanmo You, Junjie Chen, Yingyi Zhang, Xuyuan Dong, and Wenbin Zhang. Prioritizing test inputs for deep neural networks via mutation analysis. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 397–409. IEEE, 2021.

[16] Shin Yoo and Mark Harman. Regression testing minimization, selection and prioritization: a survey. *Software testing, verification and reliability*, 22(2):67–120, 2012.

[17] Jie M Zhang, Mark Harman, Lei Ma, and Yang Liu. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering*, 48(1):1–36, 2020.

[18] Chris Ziegler. A google self-driving car caused a crash for the first time. *The Verge*, 198, 2016.