ActGraph: prioritization of test cases based on deep neural network activation graph

Jinyin Chen1,2 · Jie Ge2 · Haibin Zheng1,2

Received: 15 February 2023 / Accepted: 31 July 2023 / Published online: 22 August 2023
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2023

Abstract
Widespread applications of deep neural networks (DNNs) benefit from DNN testing to guarantee their quality. In the DNN testing, numerous test cases are fed into the model to explore potential vulnerabilities, but they require expensive manual cost to check the label. Therefore, test case prioritization is proposed to solve the problem of labeling cost, e.g., surprise adequacy-based, uncertainty quantifiers-based and mutation-based prioritization methods. However, most of them suffer from limited scenarios (i.e. high confidence adversarial or false positive cases) and high time complexity. To address these challenges, we propose the concept of the activation graph from the perspective of the spatial relationship of neurons. We observe that the activation graph of cases that triggers the model’s misbehavior significantly differs from that of normal cases. Motivated by it, we design a test case prioritization method based on the activation graph, ActGraph, by extracting the high-order node feature of the activation graph for prioritization. ActGraph explains the difference between the test cases to solve the problem of scenario limitation. Without mutation operations, ActGraph is easy to implement, leading to lower time complexity. Extensive experiments on three datasets and four models demonstrate that ActGraph has the following key characteristics. (i) Effectiveness and generalizability: ActGraph shows competitive performance in all of the natural, adversarial and mixed scenarios, especially in RAUC-100 improvement (≈ ×1.40). (ii) Efficiency: ActGraph runs at less time cost (≈ ×1/50) than the state-of-the-art method. The code of ActGraph is open-sourced at https://github.com/Embed-Debugger/ActGraph.

Keywords Deep neural network · Test prioritization · Deep learning testing · Activation graph · Label

Extended author information available on the last page of the article
1 Introduction

Deep neural networks (DNNs) are widely applied in object recognition (Fel et al. 2022), image classification (Ni et al. 2022), natural language processing (Li et al. 2022), autonomous vehicles (Wang et al. 2022), etc. But it is still threatened by uncertain inputs. For example, a car in autopilot mode recognized a white truck as a cloud in the sky and caused a serious traffic accident (Danny Yadron 2016). Therefore, it is crucial to test the DNN to find vulnerabilities before deployment.

In the testing phase, numerous diverse test cases are fed into the DNN in order to evaluate the reliability of the model. These test cases require expensive manual labeling and verification. To reduce the cost of labeling, a feasible solution is to prioritize test cases to find more vulnerabilities. These test cases are more likely to expose DNN vulnerabilities are marked in advance, improving the efficiency of DNN testing.

Some test case prioritization methods have been proposed to address the labeling cost problem. They can be divided into four categories, including Neuron Coverage (NC)-based (Pei et al. 2017; Guo et al. 2018; Odena et al. 2019; Tian et al. 2018; Ma et al. 2018), Surprise Adequacy (SA)-based (Kim et al. 2019; Shen et al. 2020), Uncertainty Quantifiers (UQ)-based (Feng et al. 2020; Weiss and Tonella 2021, 2022) and mutation-based (Wang et al. 2021) test case prioritization methods. NC-based methods draw on the concept of test coverage (Zhu et al. 1997; Malaiya et al. 2002; Cadar et al. 2008) from the traditional software testing to measure the adequacy of test cases. But some studies (Li et al. 2019; Harel-Canada et al. 2020; Dong et al. 2019) have pointed out that it isn’t strongly correlated between NC and misclassified inputs, so NC cannot be effectively applied to prioritization.

UQ-based methods (Feng et al. 2020; Weiss and Tonella 2021, 2022) extract the probability distribution of test cases in the DNN confidence layer. They believe that the correctly classified cases should output higher probabilities, but misclassified cases should output multiple similar probabilities. This assumption limits their application scenarios. For example, Carlini–Wagner (C&W) (Carlini and Wagner 2017) adversarial cases can cause the DNN to output high confidence in the wrong class, so the effect of confidence-based methods will be significantly reduced.

On the other hand, SA-based prioritization methods (Kim et al. 2019; Shen et al. 2020) generally extract the hidden layer outputs of test cases. They prioritize test cases with inconsistencies based on the differences between adversarial and normal cases. However, they cannot effectively prioritize false positive (FP) cases, because the confusion of hidden layer features directly leads to the misclassification of the model.

Mutation-based test case prioritization, i.e., PRioritizing test inputs via Intelligent Mutation Analysis (PRIMA), is the state-of-the-art (SOTA) method, which uses mutation operations to make test cases as different as possible to produce different prediction results, arguing that it is easier to reveal the DNN vulnerabilities. The time complexity of PRIMA is $O(nm_1 + nm_2 N_\theta)$, where $n$ is the total
number of test cases, \( m_1 \) is the number of sample mutations, \( m_2 \) is the number of model mutations and \( N_\theta \) is the number of parameters in the model mutation. Since \( m_1, m_2 \) and \( N_\theta \) are usually very large, PRIMA has to take higher time complexity than activation-based methods (\( O(n \log n) \)).

Based on the above analysis, the existing test case prioritization methods show limitations in two aspects: (i) Limited Scenarios. UQ-based and SA-based prioritization methods have limited application scenarios. Specifically, UQ-based prioritization methods consider that misclassified cases have multiple similar probabilities, thus they are less effective for adversarial cases with high confidence. On the other hand, SA-based prioritization methods are less effective for FP cases, due to that the embedding features of FP cases are confused with the wrong classes. (ii) High Complexity. Mutation-based prioritization has a high time complexity since most of them require numerous mutations, memory read and write, and mutation query operations. Therefore, its time complexity is higher than other methods.

Therefore, to address the prioritization challenges, we explore the in-depth relationship between the test cases and the model’s dynamic features. Although the existing model activation-based prioritization methods have extracted features in the hidden layer or the confidence layer, but they overlook some more fine-grained features, such as the neuron’s activation. Therefore, we propose graph-level neuron activation features for test cases, to extract the activation graph between DNN layers. The activation graph is defined as the connection relationship of neurons. We study the differences in the activation graph for test cases. Fig. 1 shows the TSNE and activation graphs for the normal, FP, and C &W test cases, 400 each. It is LeNet-5 (LeCun et al. 1998) trained on MNIST (LeCun et al. 1998). Figure 1a, b are normal test cases with class labels 9 and 7, respectively. Figure 1c, d are FP and adversarial cases, misclassified as 9. First, from TSNE, there is a clear difference between the normal test cases (blue circle) and the other categories of test cases, especially the C &W and JSMA test cases are mostly in the right of TSNE. We only
show weighted edge values greater than 0.4, in order to show the activation graph clearly. The size of the node is determined by its node feature value. We observe the distribution differences in activation graphs for the test cases. Specifically, the edges of the last layer of the activation graph of the normal cases are only connected to the correct node, as shown in Fig. 1a, b. On the contrary, the misclassified cases will connect not only the correct node but also the wrong node in the last layer of the activation graph, as shown in Fig. 1c, d. Comparing the activation graph of normal case (Fig. 1b) and adversarial case (Fig. 1d), we observe that the similar distribution of edges in shallow layers (L1, L2), but the difference’s distribution of edges gradually increase in deeper layers (L3, L4 and L5).

Therefore, we propose a model activation graph-based test case prioritization, namely ActGraph. ActGraph regards the neurons of DNN as nodes, and the adjacency matrix as the connection relationship between nodes. Then, the node feature and the adjacency matrix are aggregated by message passing (Gilmer et al. 2017), the aggregated node feature will contain the features of neighbor nodes and the structural information between nodes, which can be effectively used for test case prioritization. ActGraph has the following two key characteristics.

(i) **Effectiveness and generalizability.** ActGraph extracts the finer-grained activation features of the test cases on the model, and converts the model activations into the spatial relationship of neurons, which can solve the limitation problem of the scene based on the model activation method. ActGraph can prioritize multiple types of test cases by learning one type of cases, which is more general than model activation-based methods.

(ii) **Efficiency.** ActGraph uses the graph to extract high-level node feature instead of mutation operations, which is much more efficient than the mutation-based approaches.

In addition, ActGraph builds a ranking model using Learning-to-Rank (L2R) (Liu 2010), which can effectively prioritize test cases by learning the node feature of test cases. According to the priority results of the ActGraph, the test cases that trigger the vulnerability can be marked earlier, thereby greatly improving the efficiency of DNN testing and effectively saving development time.

The main contributions are summarized as follows.

- Empirically, we have observed that there is a significant difference in interlayer spatial representations of DNN neurons between normal and misclassified samples, thus it can be a unique metric to distinguish them.
- Motivated by the observation, we propose a novel test case prioritization method, namely ActGraph. It extracts the spatial relationship between neuron nodes, calculates the center node feature in the activation graph, and adopts L2R model to intelligently combine center node feature to achieve efficient test case prioritization.
- Comprehensive experiments have been conducted on three datasets and four models to verify the effectiveness and efficiency of ActGraph. It outperforms the baselines in both natural and adversarial scenarios, especially in RAUC-100 (∼×
And ActGraph runs at a less time cost (∼×1/50) than the SOTA method (PRIMA).

The remainder of this paper is organized as follows. We describe the related work in Sect. 2. In Sect. 3, we describe the designed ActGraph method in detail. Experimental results are provided in Sect. 4. Threats to validity are described in Sect. 5 and conclusions are made in Sect. 6.

2 Related works

2.1 Test case prioritization for DNNs

Some DNN test case prioritization methods are proposed to solve the labeling-cost problem. According to the different dynamic features of DNNs for test cases, they can be roughly cast into four categories: NC-based (Pei et al. 2017; Guo et al. 2018; Odena et al. 2019; Tian et al. 2018; Ma et al. 2018), SA-based (Kim et al. 2019; Shen et al. 2020), UQ-based (Feng et al. 2020; Weiss and Tonella 2021) and mutation-based (Wang et al. 2021) test case prioritization methods. We compare the existing prioritization techniques with the proposed work in Table 1, which describes their application scenarios, main principles, and limitations in order to highlight our contribution.

NC-based test case prioritization NC borrows the concept of test coverage from traditional software testing to measure the adequacy of test cases. This concept was first proposed by Pei et al. (2017), who argued that DNNs with higher coverage are more secure and reliable when the input is uncertain. Ma et al. (2018) further proposed DeepGauge, which designed a set of NC metrics at multiple levels to measure the test adequacy of DNNs. However, some studies (Li et al. 2019; Harel-Canada et al. 2020; Dong et al. 2019) have pointed out that there is no strong correlation between NC and misclassified inputs, so it cannot be effectively applied to test case prioritization.

SA-based test case prioritization SA to measure the adequacy of test cases for DNN testing by calculating the difference in hidden layer features between test cases and training data, including Likelihood-based Surprise Adequacy (LSA) (Kim et al. 2019) and Distance-based Surprise Adequacy (DSA) (Kim et al. 2019). LSA refers to estimating the density of the embedding features of the test cases on the training data, while DSA defines the Euclidean distance between the embedding features of the test cases and the training data. They show that DSA is more suitable for classification models. Shen et al. (2020) proposed Multiple-Boundary Clustering and Prioritization (MCP) to uniformly select high-priority test cases from all boundary regions by dividing test cases into multiple boundary regions.
| Prioritization | Approach                  | Dataset          | Model            | B/W box | Description                                           | Limitation                   |
|----------------|---------------------------|------------------|------------------|---------|-------------------------------------------------------|------------------------------|
| NC-based       | DeepTest (Tian et al. 2018) | Driver           | LSTM, CNN       | White   | Activation value (threshold is k)                     | Ineffective                  |
|                | DeepGauge (Ma et al. 2018)   | MNIST, ImageNet  | LeNet, ResNet    | White   | NAC-k, KMNC-k, NBC-k, SNAC-k, and TKNC-k              |                              |
| SA-based       | DSA (Kim et al. 2019)       | MNIST, CIFAR-10, Driver | ConvNet, LSTM   | White   | The ratio of distances within and between classes     | High time complexity; Scenario limitation |
|                | LSA (Kim et al. 2019)       |                  |                  |         | Train a Gaussian kernel density estimator             |                              |
|                | MCP (Shen et al. 2020)      | MNIST, CIFAR-10, SVHN | ConvNet, DenseNet, RCNN | White   | Multiple decision boundary clustering                  |                              |
| UQ-based       | DeepGini (Feng et al. 2020) | MNIST, CIFAR-10, SVHN | LeNet, ResNet, VGG | Black   | Gini purity                                           | Scenario limitation          |
|                | Softmax-Entropy (Weiss and Tonella 2021) | MNist, CIFAR-10, ImageNet | LeNet, ResNet, VGG | Black   | Entropy of softmax                                    |                              |
|                | Vanilla-Softmax (Weiss and Tonella 2021) |                  |                  |         | 1-argmax(softmax)                                     |                              |
|                | PCS (Weiss and Tonella 2021) |                  |                  |         | Max(softmax)-Second(softmax)                          |                              |
| Muta-based     | PRIMA (Wang et al. 2021)    | CIFAR-100, Driver, TREC | ResNet, LSTM    | White   | Mutation input and models                              | High time and space complexity |
| Graph-based    | ActGraph (ours)             | MNIST, CIFAR-10, CIFAR-100 | LeNet, VGG, ResNet | White   | Aggregation activation and model structure             | High space complexity        |

Bold means the method proposed in this paper
**UQ-based test case prioritization**  UQ measures the likelihood of deviations between DNN predictions and known training sets. Feng et al. (2020) proposed DeepGini based on the Gini impurity, which can quickly identify test samples that may lead to DNN misclassification. Weiss et al. proposed DLS (Weiss and Tonella 2021), which contains four uncertainty metrics, which allows for zero-knowledge, transparent implementation of DNN testing. Further, Weiss and Tonella (2022) compared the prioritization effects of DeepGini and DLS, as well as the test orientation of the different methods, and found that the UQ-based methods performed similarly.

**Mutation-based test case prioritization** Wang et al. (2021) proposed PRIMA, which prioritizes those test cases that produce different prediction results through fuzzy mutation operations (including input mutation and model mutation), arguing that this is more likely to reveal DNN vulnerabilities.

In addition, some test case sampling methods have been proposed to improve the efficiency of DNN testing (Li et al. 2019; Chen et al. 2020; Ma et al. 2021; Chen et al. 2019; Li et al. 2020). These methods aim to estimate the accuracy of DNNs by sampling a few test cases. Our work aims to identify more bug-revealing test cases earlier by prioritizing test cases.

### 2.2 Graph structure of DNNs

Several studies have constructed DNNs as Graphs (DAG) to explore the performance of DNNs, including interpretability, generalization and performance analysis. (Naitzat et al. 2020) demonstrated the superiority of ReLu activation by studying the variation of the Betty number of two classes of DNN. Filan et al. (2021) proposed directly representing the fully connected layer as a weighted undirected graph, where each neuron corresponds to a node. Rieck et al. (2019) proposed neural persistence, a measure of the topological complexity of network structures that can provide a criterion for early stopping of training. They proposed “Unroll” and converting convolutional layers into graphs. Then, Vahedian et al. (2022) proposed a “Rolled” graph representation of convolutional layers to solve the DNN performance prediction problem by capturing the early DNN dynamics during the training phase. To maintain the semantic meaning of the convolutional layers, they represent each filter as a node and link the filters in successive layers with weighted edges. Zhao and Zhang (2022) proposed feature entropy to quantitatively identify individual neuron states of a specific class.

In general, most of the existing DAG studies focus on the relationship between the structural parameters of the DNN and the overall behavior of the DNN. ActGraph and the existing DAG mainly have the following differences: (i) **Graph structure.** The previous works build DNN into an undirected weighted graph, while ActGraph builds DNN as a directed weighted graph. We believe that the data flow of DNN is also directional, that is, propagation from shallow layer to deep layer, so we use directed weighted graphs. (ii) **Information of weighted edge.** The previous works
only use the weights of DNN as edges. ActGraph multiplies the neuron activation and weight edge of the model, and the graph not only contains the structural information of the model, but also expresses the dynamic information of the model. (iii) *Graph state.* The weights of DNN are not updated after training, so previous works produced a static graph. The graph generated by ActGraph is a dynamic graph, and the weight edges of the graph will change with the change of neuron activation.

### 3 Approach

#### 3.1 Overview

Existing DAG studies have been able to convert DNNs into graphs, discussing the dynamic properties of the DNN model (Filan et al. 2021; Rieck et al. 2019; Vahedian et al. 2022; Zhao and Zhang 2022). They used an undirected weighted graph to treat neurons as nodes and the weights of the model as weighted edges between nodes. However, they did not consider expressing the activation information of cases on the graph. In contrast, we use a directed weighted graph to take the model weights as the skeleton of the graph, and map the activation values to the graph, so that the features of the test case are expressed on the graph. For convenience, the definitions of symbols used in this paper are listed in Table 2.

We propose a novel test case prioritization method based on model activation graph, namely ActGraph. It consists of three stages. (i) **Test Case Activation:** the test cases are fed into the trained DNN, and each layer of the DNN outputs activation values

| Table 2 | The definitions of symbols |
|---------------------|-----------------------------|
| Symbol | Definition |
| $x$ | A test case |
| $n^l_i$ | The $i$-th neuron of the $l$-th layer |
| $F^l_i$ | The feature map of $n^l_i$ |
| $\phi^l_i$ | The average and normalized value of $F^l_i$ |
| $\theta$ | The model weights of the trained DNN |
| $W$ | The average and normalized weights of $\theta$ |
| $D = (V, E)$ | The direct weighted graph with sets of nodes and edges |
| $v_i$ | The $i$-th node of $D$ |
| $\langle v_j, v_i \rangle$ | The directed edge from $v_j$ to $v_i$ |
| $\Gamma_D(v_i)$ | The set of predecessors of $v_i$ |
| $A$ | The adjacency matrix of $D$ |
| $nf$ | The node feature |
| $cnf$ | The center node feature |
| $AGG(\cdot)$ | The aggregation function of the message passing |
| $y$ | The flag of whether the $x$ triggers the model vulnerability (0 or 1) |
| $\Omega(\cdot)$ | The regularization function |
| $T$ | The number of trees of xgboost |
(Sect. 3.2); (ii) **Feature Extraction**: activation graphs are constructed based on the activations of DNNs in each layer, and the adjacency matrix and node feature are extracted from the activation graphs. Finally, the center node feature is obtained by message passing aggregation (Sect. 3.3); (iii) **Ranking Model Building**: ActGraph adopts the framework of L2R to build a ranking model, which can utilize the center node feature, for prioritizing test cases (Sect. 3.4). It is worth emphasizing that the feature extraction proposed in this paper, that is, the weights of DNNs is extracted as directed graphs and aggregated with neuronal activation, which is significantly different from previous DAG work. In addition, ActGraph is now focused on sequential DNN models such as convolutional neural network and residual neural network. For recurrent neural network, ActGraph needs to consider more timing information by directed ring graph, which can be extended in the future. The framework of ActGraph is shown in Fig. 2.

### 3.2 Test case activation

ActGraph is a model activation-based test case prioritization method that runs during the test phase. Test cases are input to the DNN, and each layer outputs activation values. In order to facilitate the construction of the graph, the weights and activations of each neuron are averaged, and the weights and activations of each layer are normalized.

For a trained DNN and a test case \( x \). DNN has \( L \) layers, \( n^l_i \) is the \( i \)-th neuron of the \( l \)-th layer. The case \( x \) is input to the DNN, and get the output of each layer of neurons in the DNN. The activation value \( \phi^l_i \) of \( n^l_i \) is calculated as:

\[
\phi^l_i(x) = \frac{1}{\text{Height}_l \times \text{Width}_l} \sum_{\text{Height}_l \times \text{Width}_l} F^l_i(x) \tag{1}
\]

where \( F^l_i(x) \in \mathbb{R}^{\text{Height}_l \times \text{Width}_l} \) is the feature map of \( n^l_i \) output when \( x \) is input. For the convolutional layer, the output dimension is \( \text{Height}_l \times \text{Width}_l \), and the dimension of the fully connected layer is \( 1 \times 1 \).

![Fig. 2 The framework of ActGraph, which be divided into three stages: test case activation, feature extraction and ranking model building](image-url)
The neuron activation value $\varphi^l(x)$ of each layer performs the max-min normalization, which normalizes $\varphi^l(x)$ to the $[0, 1]$ range. The normalization of activation is calculated as:

$$\varphi^l(x) = \frac{\varphi^l(x) - \min(\varphi^l(x))}{\max(\varphi^l(x)) - \min(\varphi^l(x))}$$  \hspace{1cm} (2)$$

For the neuron $n^l_i$, its neuron weight is calculated as:

$$w^{l-1}_{j,i} = \frac{1}{\text{Height}_l \times \text{Width}_l} \sum_{j=1}^{\text{Height}_l} \sum_{i=1}^{\text{Width}_l} \theta^{l-1,j}_{j,i}$$ \hspace{1cm} (3)$$

where $\theta^{l-1,j}_{j,i} \in \mathbb{R}^{\text{Height}_l \times \text{Width}_l}$ represents the weight parameter between the neuron $n^l_j$ and the neuron $n^l_i$. The dimension of the neuron weight of the convolutional layer of the $l$-th layer is $\text{Height}_l \times \text{Width}_l$, and the dimension of the weight of the fully connected layer is $1 \times 1$.

Normalize the neuron weight $w^{l-1,j}$ of each layer, and convert $w^{l-1,j}$ of each layer to the range of $[0, 1]$. The normalization of weight is calculated as:

$$w^{l-1,j} = \frac{w^{l-1,j} - \min(w^{l-1,j})}{\max(w^{l-1,j}) - \min(w^{l-1,j})}$$ \hspace{1cm} (4)$$

To reduce the computational cost, ActGraph only obtains the neuron activations and their weights of the last $K$ layers of the DNN.

### 3.3 Feature extraction

In this section, we propose the steps of feature extraction for test cases in ActGraph. ActGraph extracts a set of features from the activation values of the test cases and the structural information of the model. As shown in Fig. 1, the weighted edges can significantly represent the differences of distribution between different test cases, but cannot clearly express the characteristics of neurons. Therefore, we would like to extract more effective node feature from the activation graph for prioritization. The message passing of Graph Neural Network (GNN) can aggregate the features of the current node and neighbor nodes, which is similar to the data flow of DNN, that is, the activation values of the previous layer are passed to the next layer. Specifically, we use a directed activation graph, and extract the weighted in-degrees of nodes as node feature. The weighted in-degrees are low-order node feature that represent the importance of the nodes. Further, we aggregate node feature and the adjacency matrix to obtain a higher-order node feature by message passing, namely center node feature. For the explanation of its effectiveness, we describe it in detail in Sect. 4.4.
Because the activation values of the DNN have the data flow, we construct the DNN as a directed weighted graph. Let $D = (V, E)$ be a directed weighted graph whose node set is $V$ and its edge set is $E$, where $V$ is the neuron set of DNN, and $E$ is the set of the directed weighted edges, as follows:

$$A_{j,i} = \begin{cases} 
    w_{j,i} \times \phi_i, & v_j \in \Gamma_D^{-}(v_i), v_j \notin \Gamma_D^{+}(v_i) \\
    0, & v_j \notin \Gamma_D^{-}(v_i)
\end{cases}$$

where $A$ is the adjacency matrix of $D$, $v_i$ is the $i$-th node of $D$, $w_{j,i}$ is the weight between $v_j$ and $v_i$, and $\Gamma_D^{-}(v_i) = \{ v_j \mid v_j \in V(D) \land (v_j, v_i) \in E(D) \land v_j \neq v_i \}$ is the set of predecessors of $v_i$.

We use weighted in-degree as node feature ($nf$). Degree is the simplest and most effective feature for nodes, which captures the connectivity of nodes. The $v_i$’s weight is the sum of the weights of adjacent input edges, which calculated as follows:

$$nf_i = \sum_j A_{j,i}, v_j \in \Gamma_D^{-}(v_i)$$

where $nf_i$ is the node feature value of $v_i$. Weighted in-degree is a low-order node feature. Therefore, we use the message passing of GNN to aggregate the adjacency matrix and node feature of the activation graph to obtain the center node feature ($cnf$), which is calculated as follows:

$$cnf = AGG(A, nf)$$

where the aggregation function $AGG()$ can use $Sum()$, $Max()$, and $Average()$. We use the $Sum()$ function.

![Fig. 3 The process of extract feature by ActGraph](image)
After calculating the \( cnf \) of all nodes, ActGraph only takes the \( cnf \) of the last two layers. Because we believe that the deeper activation of the model can fully express the high-dimensional characteristics of test cases. The two-layer \( cnf \) needs at least four layers of weight and activation, so we set \( K=4 \). The reason is described in Sect. 3.5.

**Example** We implement an example of three classifications on a five-layer MLP with a data set of MNIST (classes 0–2). Figure 3 shows the process by which ActGraph extracts the central node feature. For convenience, let’s set \( K = 3 \). The last three layers of the DNN have a total of 11 neurons, which are represented by yellow, green and red. ActGraph takes the last three layers of model activations and concatenates them into activation vectors. The model weights of the last three layers are represented by the adjacency matrix, and the shaded part represents the connections between the layers. ActGraph multiplies the activation vector and the weight matrix to get the activation graph. Then, ActGraph computes the in-degree of the activation graph to get the node feature. Finally, ActGraph uses message passing to aggregate the node feature with the activation graph to get the central node feature.

### 3.4 Ranking model building

In order to effectively use the central node feature to prioritize the test input, ActGraph adopts the XGBoost algorithm (Chen et al. 2016), which is an optimized distributed gradient reinforcement learning algorithm, and establishes an L2R-based ranking model for each DNN model.

The \( cnf \) of the validation set obtained by Eq. (7) is used as the training set of the ranking model, and according to the DNN’s prediction of the sample, it is labeled as 0 (prediction is correct) or 1 (prediction is wrong). The loss function for training the ranking model is as follows:

\[
obj(cnf, y) = l(y, \hat{y}) + \sum_{t=1}^{T} \Omega(f_t)
\]

where \( \hat{y} = \sum_{t=1}^{T} f_t(cnf) \), \( f_t(cnf) \) is the predicted value of the \( t \)-th tree, \( y \) is 0 or 1, \( T \) is the number of trees, and \( \Omega \) is regularization.

In summary, the process of training ranking model is shown in Algorithm 1. The input consists of a DNN \( f_1 \), the selected \( K \) layers, the data set \( X \) and label \( Y \) for training the ranking model \( f_2 \), and the central node feature \( cnf \) initialized to the empty set. The output is a trained ranking model \( f_2 \), which is used to prioritize the test cases.

During the training process of the ranking model, ActGraph reads and normalizes the weights of the \( K \) layers (Lines 1–4). Then, test cases \( x \) are successively input into model \( f_1 \) (Lines 5–15). During the iteration, ActGraph reads and normalizes the activation of neurons in \( K \) layers (Lines 6–9), where the number of neurons is the sum of the \( K \) layers. Then, ActGraph uses neuron activation to obtain the adjacency...
matrix (Line 11). Based on the adjacency matrix, ActGraph takes the in-degree of the node as the node feature $nf$ (Line 12). ActGraph aggregates node features and adjacency matrix to obtain the central node feature $cnf$ (Line 13). Finally, ActGraph uses the central node feature $cnf$ and label $Y$ to train a ranking model $f_2$ based on the L2R framework (Line 16).

Algorithm 1 ActGraph

Require: A DNN $f_1$ to be tested; The last $K$ layers selected; Validation dataset $x_c \in X = \{x_1, x_2, \ldots\}$; Label $y_c \in Y = \{y_1, y_2, \ldots\}$; $y_c = \{0, 1\}$; The set of center node feature $cnf = \{\emptyset\}$.

Ensure: A ranking model $f_2$.

1: for $l$ in $K$ do
2:   Obtain neuron weight $w^{l-1,l}$ by Eq. (3).
3:   Normalize the neuron weight $w^{l-1,l}$ by Eq. (4).
4: end for
5: for $x_c$ in $X$ do
6:   for $l$ in $K$ do
7:     Obtain neuron activation $\phi^l(x_c)$ by Eq. (1).
8:     Normalize the neuron activation $\phi^l(x_c)$ by Eq. (2).
9: end for
10: Initialize directed weighted graph $D_c$.
11: Extract Adjacency matrix $A_c$ by Eq. (5).
12: Calculate node feature $nf_c$ by Eq. (6).
13: Aggregate center node feature $cnf_c$ by Eq. (7).
14: $cnf \leftarrow cnf \cup cnf_c$.
15: end for
16: Train the ranking model $f_2$ by Eq. (8).
17: return $f_2$.

3.5 Utility analysis of center node feature

In this section, we analyze the utility of $cnf$ and explain how the $K$ value of ActGraph is determined. Let a DNN with $N$ neurons. Its output value of each layer is activated by a case $x$. The activation value $\varphi$ and the weight $W$ can be expressed as:

$$
\varphi = \begin{bmatrix} \varphi_0(x) & \varphi_1(x) & \ldots & \varphi_{N-1}(x) \end{bmatrix}_{1 \times N}
$$

(9)
where $\varphi$ and $W$ are layer normalized by Eq. (2) and Eq. (4). Then the adjacency matrix $A$ is calculated by Eq. (5) as:

$$A = \begin{bmatrix}
\varphi_0 \cdot w_{0,0} & \varphi_1 \cdot w_{0,1} & \cdots & \varphi_{N-1} \cdot w_{0,N-1} \\
\varphi_0 \cdot w_{1,0} & \varphi_1 \cdot w_{1,1} & \cdots & \varphi_{N-1} \cdot w_{1,N-1} \\
\cdots & \cdots & \cdots & \cdots \\
\varphi_0 \cdot w_{N-1,0} & \varphi_1 \cdot w_{N-1,1} & \cdots & \varphi_{N-1} \cdot w_{N-1,N-1}
\end{bmatrix}_{N \times N}$$

Then the node feature $n_f$ is calculated by Eq. (6) as:

$$n_f = \left[ \varphi_0 \sum_{j} w_{j,0} \cdots \varphi_{N-1} \sum_{j} w_{j,N-1} \right]_{1 \times N}$$

where $n_f$ is the in-degree of the activation graph. A node feature $n_f_i$ is $\varphi_i \sum_j w_{j,i}$, which indicates that the $n_f_i$ value is obtained by aggregating the activation value and input edges of $v_i$. Finally, the center node feature $cnf$ is calculated by computing $A$ and $n_f$ by Eq. (7) as:

$$cnf = \left[ \sum_{z} \varphi_z w_{z,0} n_f_z \cdots \sum_{z} \varphi_z w_{z,N-1} n_f_z \right]_{1 \times N}$$

where the $cnf_i$ of $v_i$ is $\sum_z \varphi_z w_{z,i} n_f_z$. Intuitively, $cnf_i$ is aggregated by the activation values and the $n_f$ value of neurons in the upper layer of $v_i$.

Therefore, ActGraph requires at least three layers of network to aggregate so that the $cnf$ of the last layer is valid (not zero). In the experiment, we use the last four layers of DNN, i.e. $K = 4$, to obtain the effective $cnf$ of the last two layers.

## 4 Experiment

We evaluate ActGraph through answering the following five research questions (RQs).

- **RQ1**: Does ActGraph show the SOTA prioritization performance in both natural and adversarial scenarios?
- **RQ2**: Does ActGraph show the competitive generalizability in mixed scenarios?
- **RQ3**: How to interpret ActGraph’s utility by t-SNE and heatmap visualization?
- **RQ4**: How is the stability of ActGraph under different hyperparameters (i.e. trainset size and training parameters of ActGraph)?
- **RQ5**: Is ActGraph efficient in time complexity?
4.1 Setup

**Platform** The experiments were conducted on a server equipped with Intel XEON 6240 2.6GHz X 18C (CPU), Tesla V100 32GiB (GPU), 16GiB DDR4-RECC 2666MHz (Memory), Ubuntu 16.04 (OS), Python 3.6, Keras-2.2.4, Tensorflow-gpu-1.9.0, Xgboost-1.5.2.

**Datasets** We conduct experiment on MNIST (LeCun et al. 1998), CIFAR-10 (Krizhevsky 2009) and CIFAR-100 (Krizhevsky 2009). MNIST contains 60,000 28x28 gray-scale images, and each image is marked with numbers from 0 to 9. CIFAR-10 includes 60,000 32x32 three-channel RGB-color images, which are divided into ten classes equally. CIFAR-100 includes 60,000 32x32 three-channel RGB-color images, which are divided into one hundred classes equally. For each dataset, 40,000 are used for training, 10,000 for validation and 10,000 for testing.

**Models** For MNIST, we use LeNet-5 (LeCun et al. 1998) for prioritization. On CIFAR-10, VGG16 (Simonyan and Zisserman 2015) and ResNet18 (He et al. 2016) are adopted. On even larger dataset CIFAR-100, we adopt VGG19 (Simonyan and Zisserman 2015) model. LeNet-5 has 2 convolutional layers and 3 fully connected layers. VGG16 has 13 convolutional layers and 3 fully connected layers. VGG19 has 16 convolutional layers and 3 fully connected layers. And ResNet18 has 9 residual blocks, each consisting of 2 short-circuited convolution layers. The datasets and models configurations are shown in Table 3.

**Data preparation** To verify that ActGraph is able to prioritize various test cases that trigger model bugs. We use a variety of data operations for data generation, and generate a variety of different types of datasets to make the DNN misclassify, including adversarial and natural noise cases. The natural operations include image rotation, translation and flipping, collectively referred to as Rotate. And the adversarial operations include C &W (Carlini and Wagner 2017) and Jacobian-based Saliency Map Attacks (JSMA) (Papernot et al. 2016). C &W attack leads the model to output false label with high confidence. JSMA can change only a few pixels to implement the attack, so the power of the disturbance is small.

We construct the Testset to prioritize and the Trainset for training the ranking model. For Original, Testset comes from the testset of the original dataset, and

| Datasets   | Categories | Training data | Validation data | Models     | Params       | Testing acc (%) |
|------------|------------|---------------|-----------------|------------|--------------|-----------------|
| MNIST      | 10         | 40,000        | 10,000          | LeNet-5    | 107,786      | 99.20           |
| CIFAR-10   | 10         | 40,000        | 10,000          | VGG16      | 134,326,366  | 92.04           |
| CIFAR-100  | 100        | 40,000        | 10,000          | ResNet18   | 273,066      | 92.56           |
|            |            |               |                 | VGG19      | 139,638,622  | 70.08           |
Trainset comes from the validation set of the original dataset. However, the DNN has high accuracy and only a few misclassified samples in Trainset. Therefore, these samples need to be repeatedly sampled until the balance of positive and negative samples reaches 5000 to 5000, and Testset remains unchanged. For other types of data (Rotate, JSMA, C &W and Mix), the ratio of Trainset is 5000 to 5000, that is, it consists of 5000 normally classified samples and 5000 manipulated misclassified samples, and the ratio of Testset is 8,000 to 2000. Mix is randomly sampled from four types of sets.

**Baselines** We adopt the model activation-based and mutation-based prioritization algorithms as the baselines in our experiment, including DeepGini (Feng et al. 2020), MCP (Shen et al. 2020), DSA (Kim et al. 2019) and PRIMA (Wang et al. 2021). The parameters for these algorithms are configured following their settings reported in the respective papers. In addition, in order to explore the impact of ActGraph extracted graph-level features on test case prioritization, we extracted and concatenated the confidence output and the last hidden output as our baseline, namely Act.

**Metrics** We use RAUC-n (Wang et al. 2021) as the evaluation metric for prioritization. RAUC-n reflects the prioritization effectiveness under the given number n of test cases to be labelled. RAUC-n is calculated as the ratio of the area under the curve for the test input prioritization approach to the area under the curve of the ideal prioritization, as follows:

\[
RAUC(n) = \frac{AUC(\text{real\_rank}, n)}{AUC(\text{ideal\_rank}, n)} \times 100\%
\]

where ideal\_rank is the ideal ranking result, that is, test cases with misclassification are in the first place, and real\_rank is the actual ranking result. The value range is [0, 1], and 1 indicates the best ranking result.

In our study, we consider n to be 100, 500, and 1000 respectively since the resources tend to be limited and thus the given number of test cases to be labelled tends to be small. Here, we denote their RAUC-n as RAUC-100, RAUC-500 and RAUC-1000, respectively. We also present the prioritization effectiveness on all the test cases for each subject, denoted as RAUC-ALL.

**Implementation details** Our experiments have the following settings: (1) For XGBoost ranking algorithm in ActGraph, we set Learning_rate to be 0.1, Colsample_bytree to be 0.3, and Max_depth to be 5; (2) In order to reduce the computational cost, we set K as 4 and take the cnf of the last two layers; (3) For all image data, we normalize the range of each pixel to [0, 1]. (4) All experiments run 5 times to take the average value.
4.2 Effectiveness of ActGraph

In this section, we answer RQ1 by comparing ActGraph with 5 baseline algorithms to verify its effectiveness in both natural and adversarial scenarios.

4.2.1 Prioritization result

**Implementation details** (1) Each model and dataset is set with two natural scenarios (Original and Rotate) and two adversarial scenarios (C &W and JSMA). (2) The training type is the same as the test type. (3) The size of trainset of ranking model is 2000, which contains 1000 positive samples and 1000 negative samples.

**Results and analysis** The results are shown in Table 4. The bold indicates the optimal results of different methods under the same type scenario and the same metric. In the total 64 results, ActGraph performs the best with 42 best results (65.63%), followed by DeepGini with 13 best results (20.31%) and PRIMA with 7 best results (10.94%). Then, we average the four metrics, in which ActGraph is the best, followed by PRIMA. Specifically, the average results of ActGraph are 0.865–0.939, which are 0.80–5.96% higher than PRIMA, 2.06–13.19% higher than DeepGini, 8.53–13.50% higher than Act, 18.78–21.28% higher than MCP, and 11.14–24.58% higher than DSA. In the 32 results of natural scenarios, ActGraph gets 17 best results, DeepGini gets 13 best results, and PRIMA gets 2 best results. In the 32 results of adversarial scenarios, ActGraph gets 25 best results, PRIMA gets 5 best results, MCP gets 1 best results and DSA gets 1 best results.

Because the time and cost of prioritization is limited, the number of test cases that can be labeled is often small. This also means that RAUC-100 is more important than RAUC-ALL for the test case prioritization approaches. In RAUC-100, the average result of ActGraph is 0.871, 13.19% higher than DeepGini, 5.96% higher than PRIMA, 20.20% higher than MCP, 24.36% higher than DSA, and 13.50% higher than Act. These results show that DeepGini has better effects than PRIMA in natural scenarios, PRIMA has better effects than DeepGini in adversarial scenarios, and the average results of PRIMA are better than DeepGini. ActGraph shows the SOTA effect in both adversarial and natural scenarios, especially in RAUC-100.

Then, ActGraph outperforms Act, which illustrates that cnf is more effective than model activation feature. Because cnf not only has the information of neuron activation characteristics, but also the node connection relationship between neurons. In particular, we show that ActGraph also outperforms Act in generalizability in Sect. 4.3.

ActGraph shows the best performance in most scenarios, but in some scenarios it is similar or slightly lower than baseline methods. Specifically, in CIFAR-10 & Original scenarios, DeepGini performs significantly better than ActGraph. This is because the confidence distribution of the false positive test cases matches DeepGini’s expectation. In addition, ActGraph has lower RAUC values than other baseline methods in some scenarios for CIFAR-100 & VGG19. This may be due to the
Table 4  The results of prioritization in adversarial and natural scenarios

| Datasets | MNIST | CIFAR-10 | CIFAR-100 | Average |
|----------|-------|----------|-----------|---------|
| Models   | LeNet-5 | VGG16 | ResNet18 | VGG19 |
| Type     | Original | Rotate | JSMA | CW | Original | Rotate | JSMA | CW | Original | Rotate | JSMA | CW | Original | Rotate | JSMA | CW |
| R-100 DeepGini | 0.535 | 0.967 | 0.954 | 0.906 | 0.653 | 0.794 | 0.869 | 0.639 | 0.586 | 0.845 | 0.888 | 0.559 | 0.828 | 0.822 | 0.551 | 0.430 | 0.739 |
| PRIMA | 0.386 | 0.943 | 1.000 | 0.984 | 0.530 | 0.765 | 0.892 | 0.983 | 0.415 | 0.739 | 0.889 | 0.989 | 0.886 | 0.864 | 0.766 | 0.948 | 0.811 |
| MCP | 0.279 | 0.945 | 0.870 | 0.844 | 0.369 | 0.793 | 0.931 | 0.963 | 0.489 | 0.800 | 0.917 | 0.923 | 0.400 | 0.432 | 0.379 | 0.371 | 0.669 |
| DSA | 0.160 | 0.877 | 0.854 | 0.926 | 0.428 | 0.582 | 0.712 | 0.582 | 0.262 | 0.495 | 0.684 | 0.664 | 0.586 | 0.536 | 0.834 | 0.854 | 0.627 |
| Act | 0.216 | 0.945 | 0.944 | 0.891 | 0.369 | 0.830 | 0.893 | 0.944 | 0.526 | 0.929 | 0.862 | 0.671 | 0.700 | 0.724 | 0.599 | 0.731 | 0.736 |
| ActGraph | 0.623 | 0.991 | 1.000 | 1.000 | 0.519 | 0.915 | 0.957 | 1.000 | 0.574 | 0.956 | 0.927 | 0.954 | 0.885 | 0.933 | 0.746 | 0.955 | 0.871 |
| R-500 DeepGini | 0.672 | 0.978 | 0.977 | 0.946 | 0.604 | 0.827 | 0.879 | 0.828 | 0.550 | 0.878 | 0.924 | 0.760 | 0.852 | 0.858 | 0.636 | 0.544 | 0.795 |
| PRIMA | 0.634 | 0.943 | 0.994 | 0.991 | 0.481 | 0.763 | 0.893 | 0.976 | 0.450 | 0.691 | 0.921 | 0.972 | 0.855 | 0.857 | 0.863 | 0.938 | 0.826 |
| MCP | 0.542 | 0.907 | 0.870 | 0.807 | 0.451 | 0.774 | 0.927 | 0.905 | 0.471 | 0.734 | 0.908 | 0.862 | 0.435 | 0.400 | 0.397 | 0.377 | 0.673 |
| DSA | 0.287 | 0.913 | 0.869 | 0.880 | 0.403 | 0.647 | 0.717 | 0.652 | 0.222 | 0.475 | 0.615 | 0.626 | 0.651 | 0.501 | 0.720 | 0.722 | 0.619 |
| Act | 0.293 | 0.954 | 0.942 | 0.912 | 0.451 | 0.778 | 0.896 | 0.903 | 0.456 | 0.934 | 0.826 | 0.621 | 0.807 | 0.784 | 0.673 | 0.662 | 0.743 |
| ActGraph | 0.673 | 0.973 | 0.999 | 1.000 | 0.528 | 0.852 | 0.956 | 0.990 | 0.477 | 0.976 | 0.926 | 0.970 | 0.847 | 0.865 | 0.852 | 0.946 | 0.865 |
| R-1000 DeepGini | 0.786 | 0.960 | 0.978 | 0.966 | 0.561 | 0.810 | 0.905 | 0.893 | 0.510 | 0.830 | 0.914 | 0.849 | 0.847 | 0.832 | 0.661 | 0.582 | 0.805 |
| PRIMA | 0.744 | 0.942 | 0.994 | 0.998 | 0.487 | 0.767 | 0.487 | 0.973 | 0.452 | 0.690 | 0.919 | 0.976 | 0.842 | 0.824 | 0.859 | 0.933 | 0.805 |
| MCP | 0.691 | 0.874 | 0.821 | 0.768 | 0.441 | 0.733 | 0.901 | 0.862 | 0.418 | 0.666 | 0.929 | 0.795 | 0.440 | 0.374 | 0.368 | 0.348 | 0.652 |
| DSA | 0.436 | 0.897 | 0.870 | 0.856 | 0.379 | 0.643 | 0.740 | 0.694 | 0.213 | 0.482 | 0.574 | 0.592 | 0.644 | 0.516 | 0.694 | 0.683 | 0.619 |
| Act | 0.428 | 0.964 | 0.933 | 0.901 | 0.441 | 0.753 | 0.893 | 0.904 | 0.416 | 0.916 | 0.768 | 0.613 | 0.813 | 0.797 | 0.664 | 0.620 | 0.739 |
| ActGraph | 0.769 | 0.971 | 0.996 | 0.992 | 0.521 | 0.857 | 0.943 | 0.984 | 0.458 | 0.962 | 0.916 | 0.977 | 0.835 | 0.818 | 0.882 | 0.952 | 0.865 |
Table 4 (continued)

| Datasets     | MNIST | CIFAR-10 | CIFAR-100 | Average |
|--------------|-------|----------|-----------|---------|
| Models       | LeNet-5 | VGG16   | ResNet18  | VGG19   |
| Type         | Original | Rotate | JSMA | CW | Original | Rotate | JSMA | CW | Original | Rotate | JSMA | CW | Original | Rotate | JSMA | CW | Original | Rotate | JSMA | CW | Original | Rotate | JSMA | CW |
| R-ALL DeepGini | 0.972 | 0.924 | 0.997 | 0.997 | 0.867 | 0.877 | 0.984 | 0.979 | **0.866** | 0.889 | 0.974 | 0.976 | **0.830** | 0.849 | 0.865 | 0.853 | 0.919 |
| PRIMA         | 0.963 | 0.920 | 0.999 | 0.999 | 0.854 | 0.884 | 0.976 | 0.992 | 0.857 | 0.888 | 0.980 | 0.995 | 0.824 | 0.857 | **0.933** | 0.981 | 0.931 |
| MCP           | 0.955 | 0.842 | 0.873 | 0.878 | 0.802 | 0.752 | 0.888 | 0.859 | 0.737 | 0.713 | 0.900 | 0.832 | 0.515 | 0.508 | 0.491 | 0.478 | 0.752 |
| DSA           | 0.877 | 0.936 | 0.940 | 0.928 | 0.796 | 0.832 | 0.936 | 0.937 | 0.650 | 0.746 | 0.741 | 0.737 | 0.751 | 0.779 | 0.834 | 0.826 | 0.828 |
| Act           | 0.872 | 0.957 | 0.959 | 0.948 | 0.476 | 0.874 | 0.962 | 0.968 | 0.803 | 0.937 | 0.861 | 0.819 | 0.812 | 0.853 | 0.794 | 0.770 | 0.854 |
| ActGraph      | 0.962 | **0.958** | **0.999** | **0.999** | **0.869** | **0.886** | **0.985** | **0.993** | 0.858 | **0.952** | **0.980** | **0.995** | 0.826 | **0.861** | 0.918 | **0.988** | **0.939** |

The highest RAUC value of all prioritization methods under the same experimental setting are shown in bold.

*R-N denotes RAUC-N*
large size of VGG19. ActGraph selects thousands of neurons from VGG19, much more than other models (hundreds). This adds too much useless noise and dimension to the central node feature. In the future, we plan to introduce methods such as master task neurons to eliminate useless neurons. This may improve ActGraph’s performance by increasing the proportion of valid information.

4.2.2 Guidance

**Implementation details** (1) Retrain CIFAR-10’s VGG16 and ResNet18 models using the first 100 test cases with the highest priority. (2) Set the optimizer to Stochastic Gradient Descent (SGD), learning rate to 0.001, and epoch to 5.

**Results** The experiment results are shown in Table 5. From the table, we can observe that ActGraph shows the best performance in most scenarios. In average, baseline methods improve model accuracy by 5.94% to 6.86%, while ActGraph improves accuracy by 7.35%.

**Answer to RQ1** ActGraph outperforms the baseline methods (i.e., DeepGini, PRIMA, MCP, DSA and Act) in natural and adversarial scenarios. ActGraph gets 78.13% best results in the adversarial scenarios and 53.13% best results in the natural scenarios. In RAUC-100, the average results of ActGraph are 5.96–24.36% higher than the baseline methods. In addition, ActGraph can improve model performance more effectively, with 0.49% to 1.41% higher accuracy than the baseline methods.

4.3 Generalizability of ActGraph

In the section, we find the answer to RQ2, validating the performance of ActGraph for prioritization of multiple types of test cases, especially with limited training knowledge. Limited training knowledge means that the trainset of the ranking model contains only one type of cases, which can trigger DNN vulnerabilities, but multiple types of test cases need to be prioritized in the testing phase.

**Implementation details** (1) Five training types are set up to explore the generalizability of ActGraph to Mix testset. (2) The size of trainset of ranking model is 2000, which contains 1000 positive cases and 1000 negative cases. (3) RAUC-100 and RAUC-500 are evaluated in the experiment, since the time and cost of prioritization is limited.

**Results and analysis** The results are shown in Table 6. Since DeepGini and MCP are unsupervised methods, their prioritization results are not affected by the type of testset. In the total 80 results, ActGraph shows 49 best results (61.25%), followed
| Approach | CIFAR-10 & VGG16 | | | | | CIFAR-10 & ResNet18 | | | | | Average |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Origin | Rotate | JSMA | C & W | Mix | Origin | Rotate | JSMA | C & W | Mix | Origin | Rotate | JSMA | C & W | Mix |
| Untrained | 92.03 | 80.00 | 80.00 | 80.00 | 83.22 | 92.58 | 80.00 | 80.00 | 80.00 | 83.42 | 83.13 |
| DeepGini | +0.60 | +0.89 | +13.83 | +16.64 | +7.78 | +0.31 | +1.21 | +9.19 | +10.24 | +5.72 | +5.94 |
| PRIMA | +0.41 | +0.82 | +14.22 | +17.11 | +8.03 | -0.69 | +1.32 | +9.67 | +12.23 | +5.46 | +6.86 |
| MCP | +0.00 | +1.07 | +14.18 | +16.51 | +7.17 | -3.47 | +0.80 | +8.43 | +12.10 | +4.34 | +6.11 |
| DSA | +0.37 | +1.06 | +14.01 | +16.34 | +7.66 | -1.90 | +0.67 | +8.67 | +12.47 | +4.81 | +6.42 |
| Act | +0.08 | +1.04 | +13.53 | +16.48 | +7.06 | -0.45 | +0.94 | +8.25 | +10.19 | +3.90 | +6.10 |
| ActGraph | **+0.72** | +1.35 | **+14.64** | **+17.57** | **+8.05** | +0.12 | +0.87 | **+9.89** | **+14.07** | **+6.24** | **+7.35** |
Table 6 The results of prioritization in mixed test case scenarios

| Datasets | MNIST | CIFAR-10 | ResNet18 | CIFAR-100 | Average |
|----------|-------|----------|----------|-----------|---------|
| Models   | LeNet-5 | VGG16 | ResNet18 | VGG19 |       |
| Train type | Original | Rotate | JSMA | CW | Mix | Original | Rotate | JSMA | CW | Mix | Original | Rotate | JSMA | CW | Mix |        |
| R-100 Deep-Gini | 0.899 | 0.899 | 0.899 | 0.899 | 0.738 | 0.738 | 0.738 | 0.738 | 0.828 | 0.828 | 0.828 | 0.828 | 0.828 | 0.828 | 0.828 | 0.682 | 0.682 | 0.682 | 0.682 | 0.787 |
| PRIMA | 0.944 | 0.933 | 0.984 | 0.997 | 0.957 | 0.750 | 0.745 | 0.780 | 0.912 | 0.890 | 0.736 | 0.751 | 0.839 | 0.968 | 0.897 | 0.807 | 0.776 | 0.848 | 0.852 | 0.659 | 0.851 |
| MCP | 0.964 | 0.964 | 0.964 | 0.964 | 0.858 | 0.858 | 0.858 | 0.858 | 0.853 | 0.853 | 0.853 | 0.853 | 0.853 | 0.853 | 0.853 | 0.853 | 0.479 | 0.479 | 0.479 | 0.479 | 0.789 |
| DSA | 0.930 | 0.772 | 0.789 | 0.925 | 0.833 | 0.728 | 0.508 | 0.511 | 0.662 | 0.475 | 0.376 | 0.665 | 0.432 | 0.457 | 0.457 | 0.479 | 0.541 | 0.761 | 0.837 | 0.802 | 0.671 |
| Act | 0.892 | 0.856 | 0.960 | 0.940 | 0.899 | 0.864 | 0.785 | 0.864 | 0.944 | 0.900 | 0.770 | 0.568 | 0.854 | 0.921 | 0.841 | 0.751 | 0.742 | 0.764 | 0.747 | 0.796 | 0.833 |
| Act-Graph | 0.967 | 0.979 | 1.000 | 1.000 | 0.985 | 1.000 | 0.838 | 0.905 | 0.898 | 0.970 | 0.867 | 0.853 | 0.857 | 0.978 | 0.861 | 0.849 | 0.837 | 0.765 | 0.727 | 0.818 | 0.898 |
| R-500 Deep-Gini | 0.908 | 0.908 | 0.908 | 0.908 | 0.798 | 0.798 | 0.798 | 0.798 | 0.766 | 0.766 | 0.766 | 0.766 | 0.766 | 0.766 | 0.766 | 0.766 | 0.678 | 0.678 | 0.678 | 0.678 | 0.788 |
| PRIMA | 0.903 | 0.906 | 0.979 | 0.991 | 0.929 | 0.740 | 0.726 | 0.794 | 0.910 | 0.886 | 0.744 | 0.721 | 0.831 | 0.968 | 0.887 | 0.758 | 0.779 | 0.792 | 0.847 | 0.748 | 0.842 |
| MCP | 0.909 | 0.909 | 0.909 | 0.909 | 0.773 | 0.773 | 0.777 | 0.773 | 0.732 | 0.732 | 0.732 | 0.732 | 0.732 | 0.732 | 0.732 | 0.453 | 0.453 | 0.453 | 0.453 | 0.717 |
| DSA | 0.850 | 0.799 | 0.807 | 0.839 | 0.825 | 0.618 | 0.508 | 0.590 | 0.666 | 0.532 | 0.479 | 0.538 | 0.515 | 0.455 | 0.455 | 0.424 | 0.615 | 0.678 | 0.701 | 0.689 | 0.642 |
| Act | 0.905 | 0.878 | 0.963 | 0.958 | 0.857 | 0.856 | 0.717 | 0.878 | 0.935 | 0.889 | 0.706 | 0.670 | 0.756 | 0.785 | 0.793 | 0.750 | 0.765 | 0.737 | 0.724 | 0.742 | 0.813 |
| Act-Graph | 0.965 | 0.957 | 0.986 | 0.991 | 0.971 | 0.984 | 0.869 | 0.912 | 0.944 | 0.946 | 0.786 | 0.783 | 0.862 | 0.967 | 0.852 | 0.731 | 0.833 | 0.720 | 0.728 | 0.767 | 0.878 |
| R-1000 Deep-Gini | 0.885 | 0.885 | 0.885 | 0.885 | 0.885 | 0.886 | 0.862 | 0.862 | 0.862 | 0.862 | 0.792 | 0.792 | 0.792 | 0.792 | 0.792 | 0.657 | 0.657 | 0.657 | 0.657 | 0.799 |
| PRIMA | 0.902 | 0.906 | 0.980 | 0.990 | 0.920 | 0.753 | 0.725 | 0.814 | 0.916 | 0.882 | 0.741 | 0.686 | 0.835 | 0.961 | 0.887 | 0.751 | 0.765 | 0.800 | 0.820 | 0.746 | 0.839 |
| MCP | 0.850 | 0.850 | 0.850 | 0.850 | 0.867 | 0.867 | 0.867 | 0.867 | 0.755 | 0.755 | 0.755 | 0.755 | 0.755 | 0.755 | 0.755 | 0.410 | 0.410 | 0.410 | 0.410 | 0.720 |
| DSA | 0.835 | 0.781 | 0.800 | 0.808 | 0.812 | 0.594 | 0.529 | 0.634 | 0.679 | 0.547 | 0.473 | 0.497 | 0.478 | 0.452 | 0.419 | 0.601 | 0.652 | 0.682 | 0.663 | 0.676 | 0.631 |
| Act | 0.888 | 0.857 | 0.969 | 0.969 | 0.870 | 0.823 | 0.686 | 0.879 | 0.924 | 0.880 | 0.667 | 0.651 | 0.693 | 0.715 | 0.744 | 0.724 | 0.732 | 0.716 | 0.704 | 0.726 | 0.791 |
| Act-Graph | 0.956 | 0.967 | 0.989 | 0.992 | 0.978 | 0.930 | 0.752 | 0.924 | 0.950 | 0.937 | 0.762 | 0.731 | 0.856 | 0.965 | 0.852 | 0.731 | 0.735 | 0.812 | 0.829 | 0.757 | 0.870 |
Table 6 (continued)

| Datasets | MNIST | CIFAR-10 | CIFAR-100 | VGG16 | VGG19 |
|----------|-------|----------|-----------|-------|-------|
| Models   |        |          |           |       |       |
| LeNet-5  |       |          |           |       |       |
| VGG16    |       |          |           |       |       |
| ResNet18 |       |          |           |       |       |
| ResNet18 |       |          |           |       |       |
| Train type |      | Orig-     | Rotate JSMA CW Mix | Orig- | Rotate JSMA CW Mix | Orig- | Rotate JSMA CW Mix | Orig- | Rotate JSMA CW Mix | Orig- | Rotate JSMA CW Mix | Orig- | Rotate JSMA CW Mix | Orig- | Rotate JSMA CW Mix | Orig- | Rotate JSMA CW Mix |
| R-ALL Deep-Gini | 0.973 | 0.973 | 0.973 | 0.973 | 0.973 | 0.973 | 0.973 | 0.973 | 0.973 | 0.973 | 0.973 | 0.973 | 0.973 | 0.973 | 0.973 | 0.973 | 0.973 | 0.973 |
| PRIMA | 0.908 | 0.908 | 0.908 | 0.908 | 0.908 | 0.908 | 0.908 | 0.908 | 0.908 | 0.908 | 0.908 | 0.908 | 0.908 | 0.908 | 0.908 | 0.908 | 0.908 | 0.908 |
| MCP | 0.918 | 0.918 | 0.918 | 0.918 | 0.918 | 0.918 | 0.918 | 0.918 | 0.918 | 0.918 | 0.918 | 0.918 | 0.918 | 0.918 | 0.918 | 0.918 | 0.918 | 0.918 |
| ISA | 0.934 | 0.934 | 0.934 | 0.934 | 0.934 | 0.934 | 0.934 | 0.934 | 0.934 | 0.934 | 0.934 | 0.934 | 0.934 | 0.934 | 0.934 | 0.934 | 0.934 | 0.934 |
| Act | 0.942 | 0.942 | 0.942 | 0.942 | 0.942 | 0.942 | 0.942 | 0.942 | 0.942 | 0.942 | 0.942 | 0.942 | 0.942 | 0.942 | 0.942 | 0.942 | 0.942 | 0.942 |
| Act-Graph | 0.964 | 0.964 | 0.964 | 0.964 | 0.964 | 0.964 | 0.964 | 0.964 | 0.964 | 0.964 | 0.964 | 0.964 | 0.964 | 0.964 | 0.964 | 0.964 | 0.964 | 0.964 |

The highest RAUC value of all prioritization methods under the same experimental setting are shown in bold.

RAUC denotes RAUC-N.
by PRIMA with 15 best results (18.75%), DeepGini with 7 best results and MCP with 3 best results. Then, we average the 4 metrics, and ActGraph performs the best, followed by PRIMA. The average results of ActGraph are 0.870–0.904, which are 0.953–4.65% higher than PRIMA, 2.94–11.08% higher than DeepGini, 3.83–7.95% higher than Act, 9.42–23.98% higher than DSA, and 10.90–16.09% higher than MCP. Especially, ActGraph performs better in RAUC-100, is 0.953–23.98% higher than the other baseline methods.

In the RAUC-100 of VGG16, ResNet18 and VGG19, the variance of PRIMA from 0.0062 to 0.0095, DSA from 0.0123 to 0.0163, Act from 0.0005 to 0.0184. The variance of ActGraph achieve 0.0041, 0.0028 and 0.0027, respectively. As the result, ActGraph performs more consistently than other supervised learning baseline methods. This shows the stable effectiveness of ActGraph for different types of test cases in limited training knowledge. This also shows that, for the DNN model under test, the ranking model of ActGraph does not need to be retrained frequently, because it can perform stably on different types of test cases, which indicates the generalizability of ActGraph.

Answer to RQ2 ActGraph outperforms the baseline methods (i.e., DeepGini, PRIMA, MCP, DSA and Act) in mixed scenarios. ActGraph has 61.25% of the best results, especially in average RAUC-100, which is 4.65–22.70% higher than the baseline methods. In addition, ActGraph shows better stability than the baseline methods by calculating the variance of RAUC-100. The variance of the baseline methods are 3.39 to 6.57 times that of ActGraph.

![Fig. 4](image-url) The visualization of test cases with high prioritization (i.e. FP, Rotate, JSMA and C &W). The first row is from MNIST, second row is from CIFAR-10 and third row is from CIFAR-100.
4.4 Interpretability of ActGraph

In the section, we find the answer to RQ3, explaining why ActGraph can be used effectively for prioritization. We show the visualization of test cases with high prioritization, and carry out qualitative analysis and quantitative analysis.

Implementation details  (1) The t-SNE visualization for qualitative analysis and the heat map for quantitative analysis. (2) We use mixed cases to analyze the intra-class and inter-class distances of the features of ActGraph and baseline algorithms.

Test cases visualization  The visualization of test cases with high prioritization is shown in Fig. 4. Intuitively, for the FP and Rotate test cases, they are also difficult to recognize by humans. For example, the images “7” and “5” are incomplete; The image “dog” is too bright, and the hair is too long, which blocks the basic features of dog; The colors of the images “cat”, “flatfish”, “seal” and “horse” are similar to the environment. In particular, the “tractor” and “camel” are rotated 180 degrees resulting in the blue sky at the bottom of the image, thus identifying them as “lobster” and “shark”. For JSMA and C &W, some images are also broken or blurred, such as “9” and “2” in the first line and “pine tree” in the third line. Most of the images are clear, but adding unobserved adversarial perturbations causes DNN output errors. This shows that ActGraph can pick up weak adversarial perturbations.

Qualitative analysis  The visualization of t-SNE is the first row of Fig. 5 for qualitative analysis. Intuitively, the features of Confidence and Embedding are confused, and the distances between different types are relatively close. In PRIMA, Clean and FP are close in distance, JSMA and C &W are close in distance, and Rotate is in the middle between natural and adversarial cases. This shows that although PRIMA can
distinguish between FP and adversarial cases, it is difficult to distinguish between Clean and FP cases. In ActGraph, Rotate and Clean are distinguished, and most of the FP and Rotate overlap, only a few FP and Clean intersect, and the distance between JSMA and C &W is also farther than PRIMA. This shows that the center node feature of ActGraph not only has better prioritization performance for adversarial cases, but also has better prioritization effect on natural cases and FP cases than existing methods.

**Quantitative analysis** We calculate intra-class and inter-class distances on the t-SNE for five types of cases, represented by the heatmap, for quantitative analysis, where the distance measure used Euclidean distance. The heat map is the second row of Fig. 5. For intra-class distance, ActGraph is 6.46–12.78. Except JSMA, the intra-class distance of ActGraph is smaller than other methods. For inter-class distance, intuitively, the distance between natural cases (Clean, FP and Rotate) and adversarial cases (JSMA and C &W) of ActGraph is all the farthest, with the farthest distance being 40.44, which is 1.54–2.80 times of the other three methods. This shows that ActGraph can distinguish natural samples from adversarial samples better than other methods. In addition, the distance from FP to Clean of ActGraph is 16.12, the distance from FP to Rotate is 11.77, and the difference is 4.35, while the difference of Confidence is $-0.85$, Embedding is 1.41, and PRIMA is 0.61. This shows that the FP of ActGraph is closer to Rotate and farther from Clean, and the effect of prioritization will be better. Also, ActGraph’s JSMA to C &W distance is farther than other methods.

**Answer to RQ3** ActGraph has smaller intra-class distances and larger inter-class distances than the baseline methods. For inter-class distances, the maximum inter-class distance of ActGraph is 1.54–2.80 times that of the baseline methods, and the average inter-class distance of ActGraph is 1.12–1.36 times that of the baseline methods. For inter-class distances, the minimum inter-class distance of the baseline methods are 1.52–2.04 times that of ActGraph, and the average inter-class distance are 1.28–1.47 times that of ActGraph.

**4.5 Sensitivity analysis of parameters**

In this section, we find the answer to RQ4. We analyze the influence of the size of trainset and training parameters of ActGraph.

**Implementation details** (1) We set RAUC-100 as the evaluation metric. (2) It is a VGG16 trained on CIFAR-10. (3) The type of trainset and testset are both Mix.

**Influence of trainset size** The result shows in Fig. 6. DeepGini and MCP are unsupervised methods, so are not affected by the size of the trainset. In LeNet-5 and VGG (VGG16 and VGG19), ActGraph exhibits the best prioritization performance, and the performance increases with the size of the dataset. In ResNet18,
ActGraph performs better than most baseline algorithms. Importantly, ActGraph is less affected by changes in trainset size, while the remaining three baseline algorithms (i.e. PRIMA, DSA and Act) are more affected by changes in trainset size. This shows that ActGraph can learn effective features from a small number of cases.

**Influence of parameters** We investigate the influence of main parameters in ActGraph, including three parameters (Max depth, Colsample bytree and Learning rate) in the XGBoost ranking algorithm and $K$ (unique hyperparameter of ActGraph). Figure 7 shows the effectiveness of ActGraph under different parameter settings in RAUC-100 across the four models. We find that ActGraph performs stably under different parameters of XGBoost ranking algorithm. Then, for hyperparameter $K$, ActGraph shows the best results when $K = 4$. When $K = 3$, ActGraph performance deteriorates, indicating that the aggregation of graphs requires more layers to obtain more valid information. As $K$ increases ($K > 4$), ActGraph’s performance declines, which may be too much shallow noise added to the cnf.
Answer to RQ4  For the trainset size, we set four scenarios ranging from 500 to 2000. ActGraph has stable performance and increases with the increase of the trainset size. Then ActGraph is stable for three parameters in the XGBoost ranking algorithm. However, too large a value of $K$ will cause ActGraph performance degradation. These results indicate that our previous parameter settings are appropriate.

4.6 Time complexity

In this section, we find the answer to RQ5, referring to the prioritization time cost.

Table 7  Time (seconds) taken to prioritize 10,000 test cases

| Datasets   | Models   | Methods | DeepGini | PRIMA     | Act     | MCP     | DSA     | ActGraph |
|------------|----------|---------|----------|-----------|---------|---------|---------|----------|
| MNIST      | LeNet-5  | 0.662   |          | 8049.53   | 0.642   | 0.626   | 11.59   | 103.69   |
| CIFAR-10   | VGG16    | 1.274   |          | 10864.19  | 1.115   | 1.465   | 19.09   | 246.26   |
|            | ResNet18 | 1.843   |          | 14516.53  | 1.735   | 2.444   | 37.36   | 306.44   |
| CIFAR-100  | VGG19    | 1.380   |          | 15461.26  | 1.367   | 1.705   | 19.00   | 291.05   |
**Implementation details** We measure average running time for ranking 10,000 test cases by ActGraph and baselines. We run each method for 5 times, and the average is identified as the final result.

**Results and analysis** Firstly, we theoretically analyze the complexity of ActGraph according to different steps. The time complexity consists of three parts, i.e. obtaining multi-layer activation, calculating node feature \( n_f \) and calculating center node feature \( c_{nf} \).

\[
T \sim O(t \times V) + O(t \times E) + O(t \times E) \\
\sim O(t \times E)
\]  
(15)

where \( t \) is the number of samples, \( V \) is the number of neurons, and \( E \) is the number of edges between neurons. For DL models, \( E \gg V \), so the time complexity of ActGraph is \( O(t \times E) \).

Further, we analyze the efficiency of ActGraph from the real running time. According to Table 7, the running time of ActGraph is acceptable. The time cost of ActGraph increases due to the increase in the total number of neurons and edges. In general, ActGraph is much faster than PRIMA, but is inferior to other methods. The reason is that we search more layers and neurons. Besides, ActGraph calculates the weighted edges between neurons and calculates the center node feature in the activation graph. PRIMA runs on average 50 times longer than ActGraph.

**Answer to RQ5** The average running time for ActGraph to prioritize 10,000 cases is about four minutes, which is acceptable.

**5 Threats to validity**

For the internal threat to validity, we are concerned with the correctness of implementation of our approach ActGraph, all the baseline methods and experimental scripts. To reduce this threat, we adopt the implementations of the baseline approaches released by the authors, implement ActGraph based on popular libraries, and carefully check the code of ActGraph and experimental scripts. All experiments run five times and then averaged to reduce random errors.

For the external threat to validity, we believe that it mainly lies in the subjects (i.e. models and datasets) used in our study. We use four popular DNN models and train DNN models with competitive prediction accuracy to mitigate model threats to validity. We use three popular image datasets and generate a large number of images that add natural and adversarial noise, eliminating some contingency. In the future, we will evaluate and improve our approach on datasets and models of different domains.

The construct threat to validity mainly lie in the parameters of ActGraph in our study. To reduce the threats from the parameters of ActGraph, we use the parameter settings in Sect. 4.1 and discuss the influence of the parameters in Sect. 4.6.
6 Conclusion

Aiming at the problems of limited application scenarios and high time cost of existing test case prioritization methods, we propose a test case prioritization method based on the DNN activation graph, named ActGraph. We observe that the activation graphs of cases that trigger model vulnerabilities different from those of normal cases significantly. Motivated by it, ActGraph extracts the node feature and adjacency matrix of test cases by building an activation graph, and uses the message passing mechanism to aggregate node feature and adjacency matrix to obtain more effective center node feature for test case prioritization. Extensive experiments have verified the effectiveness of ActGraph, which outperforms the SOTA method in both natural and adversarial scenarios, especially in $\text{RAUC-100} \sim \times 1.40$. And when the number of test cases is 10,000, the actual running time of the SOTA method is 50 times that of ActGraph. The experiments show that ActGraph has significantly better performance in terms of effectiveness, generalizability, and efficiency.

In the future, we will improve ActGraph’s graph construction method to adapt it to more complex models, such as transformer model and long short-term memory model. In addition, compared with SA and UQ-based methods, the time complexity of ActGraph is higher. We can select primary neurons and eliminate ineffective neurons to improve the efficiency of ActGraph.

Acknowledgements This research was supported by the National Natural Science Foundation of China (No. 62072406), the Zhejiang Provincial Natural Science Foundation (No. LDQ23F020001), the National Key Laboratory of Science and Technology on Information System Security (No. 61421110502), the National Natural Science Foundation of China (No. 62103374).

Author contributions JC: conceptualization, data curation, funding acquisition, methodology, resources, writing-review & editing, supervision, data curation, formal analysis. JG: methodology, investigation, resources, writing—original draft, software, project administration. HZ: methodology, writing—original draft, visualization, software, text polish, validation. All authors reviewed the manuscript.

Declarations

Conflict of interest The authors declare no competing interests.

References

Cadar, C., Dunbar, D., Engler, D.R.: KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs. In: Draves, R., van Renesse, R. (eds.) 8th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2008, 8–10 December, 2008, San Diego, California, USA, Proceedings, pp. 209–224. USENIX Association (2008)

Carlini, N., Wagner, D.A.: Towards evaluating the robustness of neural networks. In: 2017 IEEE symposium on security and privacy, SP 2017, San Jose, CA, USA, 22–26 May 2017, pp. 39–57. IEEE Computer Society (2017)

Chen, T., Guestrin, C.: Xgboost: A scalable tree boosting system. In: Krishnapuram, B., Shah, M., Smola, A.J., Aggarwal, C.C., Shen, D., Rastogi, R. (eds.) Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016, pp. 785–794. ACM (2016)
Chen, Y., Wang, Z., Wang, D., Yao, Y., Chen, Z.: Behavior pattern-driven test case selection for deep neural networks. In: IEEE International Conference on Artificial Intelligence Testing, AITest 2019, Newark, CA, USA, 4–9 April 2019, pp. 89–90. IEEE (2019)

Chen, J., Wu, Z., Wang, Z., You, H., Zhang, L., Yan, M.: Practical accuracy estimation for efficient deep neural network testing. ACM Trans. Softw. Eng. Methodol. 29(4), 30–13035 (2020)

Danny Yadron, D.T.: Tesla driver dies in first fatal crash while using autopilot mode (2016). https://www.theguardian.com/technology/2016/jun/30/tesla-autopilot-death-self-driving-car-elon-musk

Dong, Y., Zhang, P., Wang, J., Liu, S., Sun, J., Hao, J., Wang, X., Wang, L., Dong, J.S., Ting, D.: There is limited correlation between coverage and robustness for deep neural networks. CoRR (2019) arXiv:1911.05904

Fel, T., Rodriguez, I.F.R., Linsens, D., Serre, T.: Harmonizing the object recognition strategies of deep neural networks with humans. In: NeurIPS (2022)

Feng, Y., Shi, Q., Gao, X., Wan, J., Fang, C., Chen, Z.: Deepgini: prioritizing massive tests to enhance the robustness of deep neural networks. In: Khurshid, S., Pasareanu, C.S. (eds.) ISSTA ’20: 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, USA, 18–22 July 2020, pp. 177–188. ACM (2020)

Filan, D., Casper, S., Hod, S., Wild, C., Critch, A., Russell, S.: Clusterability in neural networks. CoRR (2021) arXiv:2103.03386

Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural message passing for quantum chemistry. In: Precup, D., Teh, Y.W. (eds.) Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6–11 August 2017. Proceedings of Machine Learning Research, vol. 70, pp. 1263–1272. PMLR (2017)

Guo, J., Jiang, Y., Zhao, Y., Chen, Q., Sun, J.: Dlfuzz: differential fuzzing testing of deep learning systems. In: Leavens, G.T., Garcia, A., Pasareanu, C.S. (eds.) 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, 04–09 November 2018, pp. 739–747. ACM (2018)

Harel-Canada, F., Wang, L., Gulzar, M.A., Gu, Q., Kim, M.: Is neuron coverage a meaningful measure for testing deep neural networks? In: Devanbu, P., Cohen, M.B., Zimmermann, T. (eds.) ESEC/ FSE’20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, 8–13 November 2020, pp. 851–862. ACM (2020)

He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, 27–30 June 2016, pp. 770–778. IEEE Computer Society (2016)

Kim, J., Feldt, R., Yoo, S.: Guiding deep learning system testing using surprise adequacy. In: Atlee, J.M., Bultan, T., Whittle, J. (eds.) Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, 25–31 May 2019, pp. 1039–1049. IEEE/ACM (2019)

Krizhevsky, A.: Learning multiple layers of features from tiny images. Master’s thesis, University of Tront (2009)

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE 86(11), 2278–2324 (1998)

Li, Z., Ma, X., Xu, C., Cao, C., Xu, J., Lü, J.: Boosting operational DNN testing efficiency through conditioning. In: Dumas, M., Pfahl, D., Apel, S., Russo, A. (eds.) 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, 26–30 August 2019, pp. 499–509. ACM (2019)

Li, Z., Ma, X., Xu, C., Cao, C.: Structural coverage criteria for neural networks could be misleading. In: Sarma, A., Murta, L. (eds.) Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results, ICSE (NIER) 2019, Montreal, QC, Canada, 29–31 May 2019, pp. 89–92. IEEE/ACM (2019)

Li, Z., Ma, X., Xu, C., Xu, J., Cao, C., Lu, J.: Operational calibration: debugging confidence errors for DNNs in the field. In: Devanbu, P., Cohen, M.B., Zimmermann, T. (eds.) ESEC/FSE’20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, 8–13 November 2020, pp. 901–913. ACM (2020)

Li, S., Dong, T., Zhao, B.Z.H., Xue, M., Du, S., Zhu, H.: Backdoors against natural language processing: a review. IEEE Secur. Priv. 20(5), 50–59 (2022)
Liu, T.: Learning to rank for information retrieval. In: Crestani, F., Marchand-Maillet, S., Chen, H., Efthimiadis, E.N., Savoy, J. (eds.) Proceeding of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2010, Geneva, Switzerland, 19–23 July 2010, p. 904. ACM

Ma, L., Juefei-Xu, F., Zhang, F., Sun, J., Xue, M., Li, B., Chen, C., Su, T., Li, L., Liu, Y., Zhao, J., Wang, Y.: Deepprogue: multi-granularity testing criteria for deep learning systems. In: Huchard, M., Kästner, C., Fraser, G. (eds.) Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, 3–7 September 2018, pp. 120–131. ACM (2018)

Ma, W., Papadakis, M., Tsakmalis, A., Cordy, M., Traon, Y.L.: Test selection for deep learning systems. ACM Trans. Softw. Eng. Methodol. 30(2), 13–11322 (2021)

Malaiya, Y.K., Li, M.N., Bieman, J.M., Karcich, R.: Software reliability growth with test coverage. IEEE Trans. Reliab. 51(4), 420–426 (2002)

Naitzat, G., Zhitnikov, A., Lim, L.: Topology of deep neural networks. J. Mach. Learn. Res. 21, 184–118440 (2020)

Ni, J., Xiang, D., Lin, Z., López-Martínez, C., Hu, W., Zhang, F.: DNN-based polsar image classification on noisy labels. IEEE J. Sel. Top. Appl. Earth Obs. Remote. Sens. 15, 3697–3713 (2022)

Odena, A., Olsson, C., Andersen, D.G., Goodfellow, I.J.: Tensorfuzz: Debugging neural networks with coverage-guided fuzzing. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9–15 June 2019, Long Beach, California, USA. Proceedings of Machine Learning Research, vol. 97, pp. 4901–4911. PMLR (2019)

Papernot, N., McDaniel, P.D., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A.: The limitations of deep learning in adversarial settings. In: IEEE European Symposium on Security and Privacy, EuroS &P 2016, Saarbrücken, 21–24 March 2016, pp. 372–387. IEEE (2016)

Pei, K., Cao, Y., Yang, J., Jana, S.: Deeptest: automated whitebox testing of deep learning systems. In: Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, 28–31 October 2017, pp. 1–18. ACM (2017)

Rieck, B., Togninalli, M., Bock, C., Moor, M., Horn, M., Gumbusch, T., Borgwardt, K.M.: Neural persistence: A complexity measure for deep neural networks using algebraic topology. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, 6–9 May 2019. OpenReview.net

Shen, W., Li, Y., Chen, L., Han, Y., Zhou, Y., Xu, B.: Multiple-boundary clustering and prioritization to promote neural network retraining. In: 35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, Melbourne, Australia, 21–25 September 2020, pp. 410–422. IEEE (2020)

Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7–9, 2015, Conference Track Proceedings

Tian, Y., Pei, K., Jana, S., Ray, B.: Deeptest: automated testing of deep-neural-network-driven autonomous cars. In: Chaudron, M., Crnkovic, I., Chechik, M., Harman, M. (eds.) Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27–June 03 2018, pp. 303–304. ACM (2018)

Vaheidian, F., Li, R., Trivedi, P., Jin, D., Koutra, D.: Leveraging the graph structure of neural network training dynamics. In: Hasan, M.A., Xiong, L. (eds.) Proceedings of the 31st ACM International Conference on Information & Knowledge Management, Atlanta, GA, USA, 17–21 October 2022, pp. 4545–4549. ACM (2022)

Wang, Z., You, H., Chen, J., Zhang, Y., Dong, X., Zhang, W.: Prioritizing test inputs for deep neural networks via mutation analysis. In: 43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22–30 May 2021, pp. 397–409. IEEE (2021)

Wang, J., Su, W., Luo, C., Chen, J., Song, H., Li, J.: CSG: classifier-aware defense strategy based on compressive sensing and generative networks for visual recognition in autonomous vehicle systems. IEEE Trans. Intell. Transp. Syst. 23(7), 9543–9553 (2022)

Weiss, M., Tonella, P.: Fail-safe execution of deep learning based systems through uncertainty monitoring. In: 14th IEEE Conference on Software Testing, Verification and Validation, ICST 2021, Porto de Galinhas, Brazil, 12-16 April 2021, pp. 24–35. IEEE (2021)

Weiss, M., Tonella, P.: Simple techniques work surprisingly well for neural network test prioritization and active learning (replicability study). In: Ryu, S., Smaragdakis, Y. (eds.) ISSTA’22: 31st ACM
SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, South Korea, 18–22 July 2022, pp. 139–150. ACM (2022)
Zhao, Y., Zhang, H.: Quantitative performance assessment of CNN units via topological entropy calculation. In: The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, 25–29 April 2022. OpenReview.net
Zhu, H., Hall, P.A.V., May, J.H.R.: Software unit test coverage and adequacy. ACM Comput. Surv. 29(4), 366–427 (1997)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.

Authors and Affiliations

Jinyin Chen1,2 · Jie Ge2 · Haibin Zheng1,2

Haibin Zheng
haibinzheng320@gmail.com

Jinyin Chen
chenjinyin@zjut.edu.cn

Jie Ge
2112103116@zjut.edu.com

1 Institute of Cyberspace Security, Zhejiang University of Technology, Hangzhou 310023, Zhejiang, China

2 College of Information Engineering, Zhejiang University of Technology, Hangzhou 310023, Zhejiang, China