DeepCon: Contribution Coverage Testing for Deep Learning Systems

Zhiyang Zhou[1,2], Wensheng Dou[1,2,3], Jie Liu[1,2,3], Chenxin Zhang[1,2], Jun Wei[1,2], Dan Ye[1,2]
[1]*State Key Lab of Computer Sciences, Institute of Software, Chinese Academy of Sciences, Beijing, China*
[2]*University of Chinese Academy of Sciences, Beijing, China*
[3]*Institute of Software Technology, Chinese Academy of Sciences, Nanjing, China*
{zhouzhiyang18, wsdou, ljie, zhangchenxin17, wj, yedan}@otcaix.iscas.ac.cn

***Abstract*—Deep learning (DL) has been widely adopted in many safety-critical scenarios. Deep neural networks (DNNs) usually play the core part in these DL systems. Existing studies have shown that DNNs can suffer from various vulnerabilities, and cause severe consequences. To improve the testing adequacy of DNNs, researchers have proposed several coverage criteria, e.g., neuron coverage in DeepXplore. The prediction result of a DNN is jointly determined by the outputs of neurons and the connection weights that they connect into next-level neurons. However, existing coverage criteria use only the output of a neuron to determine the activation state of the neuron and ignore the connection weights it emits.**

In this paper, we propose *DeepCon*, a novel contribution coverage. In DeepCon, we define a term *contribution* as the combination of the output of a neuron and the connection weight it emits, and use the contribution coverage to gauge the testing adequacy of DNNs. DeepCon can thoroughly cover both neurons and the connection weights they emit and can scale well to large DNNs. We further propose a contribution coverage guided test generation approach, *DeepCon-Gen*, which can automatically generate tests and activate inactivated contributions of DNNs. We evaluate DeepCon and DeepCon-Gen on five different DNNs over two popular datasets. The experimental results show that DeepCon can well present the testing adequacy of these DNNs. DeepCon-Gen can effectively activate the inactivated contributions, and 62.6% of the generated tests can lead to mispredictions.

***Index Terms*—Deep learning, deep neural network, coverage criteria, contribution coverage, adversarial example detection**

I. Introduction

Benefiting from large dataset, deep learning (DL) systems have achieved great success in various applications in the last decade, e.g., computer vision [1], [2], speech recognition [3], natural language processing [4], and autonomous driving [5]. Deep neural networks (DNNs) play a key part in these DL systems. However, DNN-based deep learning systems can sometimes exhibit erroneous behaviors, and cause severe consequences. For example, unexpected driving conditions in the Google and Uber autonomous vehicles have caused several car accidents [6], [7].

A DNN consists of a group of interconnected neurons, which are organized as layers. Typically, a DNN contains the input layer, one or multiple hidden layers, and the output layer. Figure 1 shows a four-layer fully-connected DNN example. In a typical fully-connected DNN, each neuron computes its output by applying an activation function to its inputs. Each neuron is fully connected with all neurons in the next layer, and each edge has a weight, which indicates the strength of the connection. In this paper, we focus on stateless DNNs, e.g, CNN [8], [9] and its variants.

A DNN can be considered as a function that transforms an input to its output by aggregating all neurons with different connection weights, all of which can contribute to the final prediction result. This computation logic (i.e., connection weights) is learned during the training phase and is unclear to developers. Therefore, different from traditional programs, it is challenging to thoroughly test a DNN. To systematically and thoroughly test DNNs, several testing adequacy criteria have been proposed. DeepXplore [10] proposed a neuron coverage (DNC) criterion to measure how many unique neurons have been activated by test inputs. DeepGauge [11] proposed a group of k-multi-granularity neuron coverage (KMNC), in which the output of a neuron is partitioned into k sections. KMNC measures how many unique sections have been covered by test inputs. DeepCover [12] proposed a MC/DC coverage criterion based on the condition-decision dependencies between adjacent neurons.

These coverage criteria [10], [11] use only the output of a neuron to determine its activation state and ignore the connection weights it emits. As the importance of a neuron to the next-level neuron is also greatly affected by the connection weights, DeepCover used the outputs of adjacent neurons to test the independent influence of the shallower neuron (denoted as *condition neuron*) on the deeper neuron (denoted as *decision neuron*). However, this combinatorial coverage of neurons fails to thoroughly cover the connection weights since they still use only the output of the *decision neuron* to determine its activation state and ignore the connection weights it emits. Moreover, to test the independent influence of a condition neuron on the decision neuron, DeepCover needs to freeze the outputs of all other neurons in the same layers as these two neurons, and cannot easily scale to large and complex DNNs [11], [13].

Mathematically, the prediction result of a DNN is jointly decided by the outputs of neurons and the connection weights they emit. In this paper, we use *contribution* to denote the combination (or product) of the output of a neuron and the connection weight it emits, and use contribution instead of neuron as the basic logic unit of a DNN to gauge the testing

adequacy of the DNN. We first propose *DeepCon*, a novel contribution coverage criterion. DeepCon can naturally and thoroughly cover both the outputs of neurons and the connection weights they emit. Since the number of contributions of a DNN is equal to the number of connection weights of the DNN, DeepCon can scale well to large and complex DNNs. Benefiting from the simultaneous coverage of neurons and connection weights, DeepCon can better present the testing thoroughness of DNNs and has a better performance in some closely related tasks such as adversarial example detection [14], [15]. We then develop a contribution coverage guided test generation approach, *DeepCon-Gen*. In DeepCon-Gen, We transform the test generation into an optimization problem, and construct a joint-objective optimization function for an inactivated contribution. For scalability, we solve this function based on a gradient-ascent approach [16], [17].

We evaluate DeepCon and DeepCon-Gen on three small DNNs (i.e., LeNet-1, LeNet-4 and LeNet-5 [8]) and two large DNNs (i.e., VGG-19 [18] and ResNet-50 [19]) over two popular datasets (i.e., MNIST [20] and ImageNet [21]), respectively. The experiments show that contribution criterion can scale well to different sized DNNs and can well present the testing adequacy of these DNNs. For example, the contribution coverage of any one of these DNNs grows with the increase of test inputs and does not reach 100% even on all test inputs. This indicates that we need more tests to thoroughly test these DNNs if we use contribution coverage as a testing criterion. More importantly, when compared with existing coverage criteria in another closely related application task, i.e., adversarial example detection, DeepCon can obtain a higher AUC [22] of 13.00%. Lastly, the experimental results of DeepCon-Gen shows that DeepCon-Gen can effectively activate inactivated contributions, and 62.60% of the generated tests can lead to the mispredictions of DNNs.

In summary, we make the following main contributions.

- We propose a new contribution coverage criteria, DeepCon, which first explicitly combines the output of a neuron and the connection weight it connects to the next-level neuron.
- We develop a contribution coverage guided test generation approach, DeepCon-Gen, which adopts the gradient-ascent approach and transforms the inactivated contribution activation into an optimization problem.
- We evaluate DeepCon and DeepCon-Gen in real-world DNNs and datasets. The experimental results show that DeepCon can well present the testing adequacy of DNNs, and DeepCon-Gen can effectively activate inactivated contributions.

II. Preliminaries and Motivation

In this section, we introduce the deep neuron network and the limitations of existing coverage criteria, i.e., neuron-output-based coverage criteria.

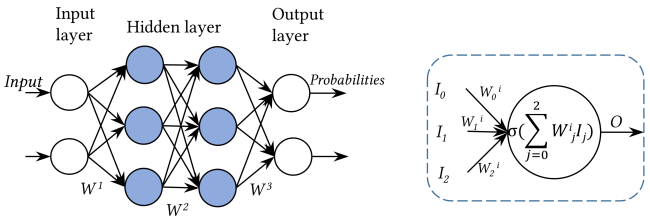


Fig. 1. A four-layer fully-connected deep neuron network.

A. Deep Neural Network

Deep Neural Network (DNN) is a typical layer-by-layer feature extraction machine learning algorithm. In recent years, due to a large number of dataset available [20], [21], specific hardware [23], [24] and software [25]–[27], and more efficient training methods, DNNs have shown much higher generalization performance than other machine learning algorithms or even human beings [10].

A DNN is a complex and high dimensional function with a large number of neurons and trainable connection weights, it accepts an input data and output its prediction label. Generally, a DNN consist of three kinds of layers, i.e., the input layer, one or multiple hidden layers, and the output layer. Each non-input layer contains a large number of neurons and trainable connection weights. A simplified four-layer fully-connected DNN architecture is shown in Figure 1, which consists of one input layer, two hidden layers and one output layer. In this DNN, each non-output neuron is fully connected with all neurons in the next layer, and each edge has a weight (i.e., trainable parameter). We formalize a DNN as follows.

In a DNN, let l be the number of layers in the DNN, s_i be the number of neurons in the i-th layer, and $n_{i,j}$ be the j-th neuron in the i-th layer. We use $w^i_{h,j}$ to denote the connection weight between $n_{i\text{-}1,h}$ and $n_{i,j}$, and W^i=$\{w^i_{h,j}|0 \leq h < s_{i\text{-}1}, 0 \leq j < s_i\}$ to denote the weight set in the i-th layer L_i. For example, $n_{1,3}$ represents the 3-th neuron in the 1-th layer, and $w^1_{2,3}$ represents the connection weight between neuron $n_{0,2}$ and neuron $n_{1,3}$. We use a tuple $G = (N, W)$ to denote a DNN, where $N = \{n_{i,j}|0 \leq i < l, 0 \leq j < s_i\}$, and $W = \{W^i|1 \leq i < l\}$.

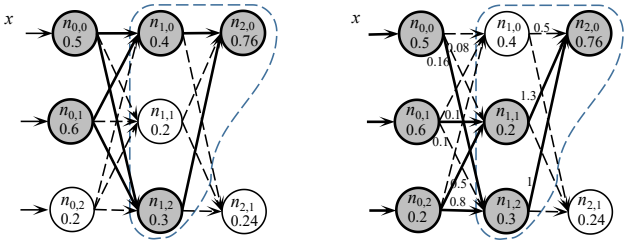
A neuron is the basic computing unit in a DNN. Let $o_{i,j}$ be the output of neuron $n_{i,j}$. The computation process of $o_{i,j}$ can be decomposed into Equation (1), (2) and (3).

$$u^i_{h,j} = w^i_{h,j} \cdot o_{i\text{-}1,h} \tag{1}$$

$$U^i_j = b_{i,j} + \sum_{h=0}^{s_i\text{-}1} u^i_{h,j} \tag{2}$$

$$o_{i,j} = \sigma(U^i_j) \tag{3}$$

In Equation (1), $u^i_{h,j}$ denotes an input neuron $n_{i\text{-}1,h}$'s contribution to neuron $n_{i,j}$, which is actually the product of the output $o_{i\text{-}1,h}$ of $n_{i\text{-}1,h}$ and the connection weight $w^i_{h,j}$ that is emitted by $n_{i\text{-}1,h}$. In Equation (2) and (3), $b_{i,j}$ denotes the bias



(a) NOB coverage criteria (b) DeepCon

Fig. 2. Comparison of NOB coverage criteria and DeepCon. The text and number in each node represent the ID and output of the neuron, respectively. In Figure 2(a) and Figure 2(b), nodes with gray background represent activated neurons extracted by NOB coverage criteria (with $t > 0.25$) and DeepCon (with $t > 0.25$), respectively.

parameter for neuron $n_{i,j}$, and U_j^i denotes the contributions form previous neurons $\{n_{i\text{-}1,h}|0 \leq h < s_{i\text{-}1}\}$ to neuron $n_{i,j}$. The activation function σ on U_j^i greatly increase the nonlinearity of neuron $n_{i,j}$. Various activation functions can be applied, e.g., ReLU [28] for hidden layers, and softmax [28]–[30] for output layers.

In the training phase, $w_{h,j}^i$ and $b_{i,j}$ in Equation (1) and (2) are learned by using an efficient optimization algorithm, e.g., back-propagation [31] gradient descent or its variants [32]–[34]. In the prediction phase, $w_{h,j}^i$ and $b_{i,j}$ are used to calculate the probability of each category. Generally, the category with the highest probability is used as the prediction result, i.e., $label = \arg\max_j o_{l\text{-}1,j}$, where $0 \leq j < s_{l\text{-}1}$.

We can see that a DNN is a typical layer-by-layer network, which transforms input data into output categories. During this transformation process, the outputs of neurons and the connection weights they connect to next-level neurons together determine the final prediction result. In other words, the importance of each non-output layer neuron should be determined by its contribution to the prediction result.

B. Limitations of Neuron-Output-Based Coverage Criteria

To evaluate how thoroughly a DNN has been tested, researchers have proposed several coverage criteria for DNNs. As a pioneer, DeepXplore [10] proposed neuron coverage, in which, if the output of a neuron (e.g., $o_{i,j}$ in Equation (3)) is larger than a given threshold, the neuron is considered to be activated. DeepXplore inspires a group of studies, e.g., k-multi-granularity neuron coverage (KMNC) [11], MC/DC coverage [12], and t-way (t neurons in the same layer) combinational coverage [35]. Note that, all these coverage criteria only use the output of a neuron to determine the neuron's state. We summary these coverage criteria as *neuron-output-based (NOB) coverage criteria*.

The basic intuition behind NOB coverage criteria is that, the larger a neuron's output, the stronger its influence on its following neurons [10]. However, for different neurons in the same layer, this intuition may not always be true. Since the connections emitted by different neurons have different weights, a neuron with a larger output but a smaller weight may contribute less to the same neuron downstream than another neuron with a smaller output but a larger weight. If we only consider the output of a neuron, it is hard to thoroughly test the behaviors of a DNN. Let us suppose $n_{l\text{-}1,j}$ represents the prediction result's corresponding neuron in output layer of DNN for a given input x and $o(n, x)$ represents the output of neuron n on x. For any two penultimate layer neurons $n_{l\text{-}2,h}$ and $n_{l\text{-}2,h+k}$, let $c = \frac{o(n_{l\text{-}2,h},x)}{o(n_{l\text{-}2,h+k},x)}$. If $o(n_{l\text{-}2,h}, x) > o(n_{l\text{-}2,h+k}, x)$, we cannot say that $u_{h,j}^{l\text{-}1} > u_{h+k,j}^{l\text{-}1}$, i.e., if we only know that $n_{l\text{-}2,h}$'s output is larger than $n_{l\text{-}2,h+k}$'s, we cannot say that $n_{l\text{-}2,h}$ makes greater contribution to the prediction result than that of $n_{l\text{-}2,h+k}$. Only when $w_{h,j}^{l\text{-}1} > \frac{w_{h+k,j}^{l\text{-}1}}{c}$ is also satisfied, can we say that $n_{l\text{-}2,h}$ makes greater contribution to the prediction result than that of $n_{l\text{-}2,h+k}$.

Take the simplified DNN shown in Figure 2 as an example. The DNN takes x as input and classifies it into two categories, represented by neuron $n_{2,0}$ or $n_{2,1}$, respectively. Suppose $n_{2,0}$ is the predicated result neuron for a given input x. As shown in Figure 2(a), if we take the activation threshold as 0.25 in the NOB coverage criteria, $n_{1,0}$ and $n_{1,2}$ will be considered as activated, $n_{1,1}$ is considered as inactivated. However, as shown in Figure 2(b), the contributions (i.e., $u_{h,j}^i$ in Equation (1)) from $n_{1,0}$ and $n_{1,1}$ to $n_{2,0}$ are 0.2 (i.e., $u_{0,0}^2 = 0.4 * 0.5 = 0.2$) and 0.26 (i.e., $u_{1,0}^2 = 0.2 * 1.3 = 0.26$) respectively, thus $n_{1,1}$ is more important than $n_{1,0}$.

From the above analysis, we can see that the NOB coverage criteria may introduce some *misfitting* to the coverage testing to some degree. Some combinatorial coverage methods such as [12] proposed to use the outputs of adjacent neurons (i.e., condition neurons and decision neurons) to cover the connection weights between them. However, they still use only the outputs of neurons to determine the activation states of both decision neurons and decision neurons, thus fails to eliminate this *misfitting* and cannot cover the connection weights emitted by decision neurons. On the other hand, due to the huge combination space of multiple neurons in large and complex DNNs, using the combination of multiple neurons to gauge the testing adequacy may lead to over-high or even unacceptable computational costs. For example, to test the independent influence of a condition neuron on its decision neuron, [12] needs to freeze the outputs of all other neurons in the same layers of these two adjacent neurons. As we all know that, for a large DNN, the output of a decision neuron is jointly determined by the outputs of a large number of previous neurons and the connection weights that connect into this neuron, it is difficult for us to test the independent influence of the conditional neuron on the decision neuron.

In summary, no coverage criterion can simultaneously cover both the outputs of neurons and the connection weights they emit, and scale well to large DNNs.

III. Contribution Coverage Criteria

To thoroughly cover both the outputs of neurons and the connection weights they connect to next layers, we introduce

a term *contribution* to present the internal logic coverage of a DNN. In the subsection, we first introduce the concept of contribution and how to extract activated contributions. Then, we put forward the definition of contribution coverage criterion for DNNs.

A. Contribution Extraction

In a DNN, we use $c^i_{h,j}$ to denote the connection from $n_{i\text{-}1,h}$ to $n_{i,j}$, and use $u^i_{h,j} = (n_{i\text{-}1,h}, c^i_{h,j})$ to denote the contribution of $n_{i\text{-}1,h}$ to $n_{i,j}$. Thus, for a given input x, $u^i_{h,j}(x) = w^i_{h,j} \cdot o_{i\text{-}1,h}(x)$, in which $o_{i\text{-}1,h}(x)$ denotes the output of $n_{i\text{-}1,h}$ on x, and $w^i_{h,j}$ denotes the weight of connection $c^i_{h,j}$. We use $U^i_j(x) = \{u^i_{h,j}(x) | 0 \leq h < s_{i\text{-}1}\}$ to denote the set of contributions whose connections end with $n_{i,j}$.

Note that, the value range of the contributions in different $U^i_j(x)$ may vary largely. Thus, we normalize $u^i_{h,j}(x)$ into a value within [0,1]. We use $nu^i_{h,j}(x)$ to denote the normalized contribution of $u^i_{h,j}(x)$, and use Equation (4) to normalize $u^i_{h,j}(x)$. In this computation, we exclude the zero contributions because the zero contribution means that the previous neuron have no influence on the following neuron.

$$nu^i_{h,j}(x) = \frac{u^i_{h,j}(x) - min(U^i_j(x))}{max(U^i_j(x)) - min(U^i_j(x))} \tag{4}$$

Definition 1 (Contribution activation): Given an input x, if a contribution $u^i_{h,j}$'s normalized contribution $nu^i_{h,j}(x)$ is greater than a threshold value t, and its output neuron $n_{i,j}$ is activated by x, then this contribution is activated by input x. We use $isAct(u^i_{h,j}, t, x)$ to denote whether a contribution is activated or not.

We say a neuron $n_{i,j}$ is activated by input x, if there exists a contribution $u^{i+1}_{j,\cdot}(x) = w^{i+1}_{j,\cdot} \cdot o_{i,j}$ that is activated by input x. Note that, for the neurons in the output layer, we only consider the neuron that has the largest output as activated, and all other neurons in the output layer are considered as inactivated.

Back-propagation Contribution Extract Algorithm. For a given DNN and input x, the back-propagation contribution extract algorithm extracts activated contributions back from the output layer until the input layer. Algorithm 1 shows how it works. We use $\widetilde{U}^i$ to represent the set of activated contributions from the $(i-1)$-th layer to the i-th layer (i.e., the i-th layer's contribution set) and use $\widetilde{U}^i.InputNeurons$ to represent the input neurons of $\widetilde{U}^i$.

First, we check the output layer (i.e., the $(l-1)$-th layer), and consider the neuron with the largest output as activated (Line 1).

Second, for each activated neuron $n_{i,j}$ in the current layer (Line 2 and 5), we use $getActivatedContributions(n_{i,j}, t, x)$ to extract the activated contribution set $\widetilde{U}^i_j$ whose connection ends with $n_{i,j}$ (Line 2 and 6) and aggregate $\widetilde{U}^i_j$ to $\widetilde{U}^i$ (Line 7). In the implementation of $getActivatedContributions$, we use $isAct$ (defined in Definition 1) to check the state of each contribution $u^i_{h,j}$. Note that, if $u^i_{h,j}$ is activated, we further consider its input neuron $n_{i\text{-}1,h}$ and connection $c^i_{h,j}$ as activated.

Algorithm 1: Back-propagation contribution extract algorithm

Input: x (test input), t (threshold)
Output: $\widetilde{U}(x)$ (all activated contributions)

1 $n_{l\text{-}1,j} = F(x)$
2 $\widetilde{U}^{l\text{-}1}(x) = getActivatedContributions(n_{l\text{-}1,j}, t, x)$
/*get activated contribution set whose connection ends with neuron $n_{l\text{-}1,j}$ by using $isAct(u^{l\text{-}1}_{\cdot,j}, t, x)$ */
3 aggregate $\widetilde{U}^{l\text{-}1}(x)$ to $\widetilde{U}(x)$
4 **for** *layer i back from l-2 to 1* **do**
5 **for** *neuron $n_{i,j}$ in $\widetilde{U}^{i+1}$.InputNeurons* **do**
6 $\widetilde{U}^i_j(x) = getActivatedContributions(n_{i,j}, t, x)$
7 aggregate $\widetilde{U}^i_j(x)$ to $\widetilde{U}^i(x)$
8 aggregate $\widetilde{U}^i(x)$ to $\widetilde{U}(x)$
9 return $\widetilde{U}(x)$

Third, we aggregate the activated contribution set $\widetilde{U}^i(x)$ of the current layer to $\widetilde{U}(x)$ (Line 8), and move back from the current layer to the previous layer and continue to perform the second and third steps until the input layer (Line 4-8).

Take the simplified DNN in Figure 2(b) as an example. For a given x, in the output layer, neuron $n_{2,0}$ has the largest output, and is considered as activated. If we set the contribution threshold t as 0.25, the contributions activated by input x are $u^2_{1,0}$ and $u^2_{2,0}$, i.e., $(n_{1,1}, c^2_{1,0})$ and $(n_{1,2}, c^2_{2,0})$. Since the normalized contribution $nu^2_{0,0}$ from $n_{1,0}$ to $n_{2,0}$ is not larger than 0.25, we can know that neuron $n_{1,0}$ is not activated. Further, from activated neurons $\widetilde{U}^2.InputNeurons = \{n_{1,1}, n_{1,2}\}$, we can know that contributions $(n_{0,0}, c^1_{0,2})$, $(n_{0,1}, c^1_{1,1})$, $(n_{0,2}, c^1_{2,1})$ and $(n_{0,2}, c^1_{2,2})$ are activated by x.

We can see that, the contribution computation cost of a DNN on a given input is approximately equal to the prediction cost of the DNN on the input, and the corresponding extraction cost of activated contributions is linear. Therefore, the the back-propagation contribution extract algorithm can scale well to large DNNs.

B. Contribution Coverage

Contribution coverage presents the ratio of unique contributions activated by a group of test inputs and all contributions in a DNN. Let $T = \{x_1, x_2, x_3, ...\}$ be the test inputs, and U be the set for all contributions in a DNN. The contribution coverage (**ConC**) of a DNN for all test inputs T can be described as follows.

$$ConC(T) = \frac{|u | \exists x \in T, u \in U, isAct(u, t, x)|}{|U|} \tag{5}$$

Take the DNN in Figure 2(b) as an example, for the input x and the contribution threshold 0.25, the ConC of the DNN is 6/15 = 40%.

In a DNN, some layers (e.g., shallower layers that extract low-dimensional features) can be easily covered, while other

layers (e.g., deeper layers that extract high-dimensional features) cannot be easily covered [36], [37]. We further propose contribution coverage of each layer in a DNN. Let U^i be the set for all contributions the i-th layer of a DNN. The contribution coverage of the i-th layer can be described as follows.

$$ConC_i(T) = \frac{|u|\exists x \in T, u \in U^i, isAct(u,t,x)|}{|U^i|} \quad (6)$$

Take the DNN in Figure 2(b) as an example, for the input x and the contribution threshold 0.25, the ConC_1 and ConC_2 will be 4/9 $\approx$ 44% and 2/6 $\approx$ 33%, respectively.

IV. Contribution Guided Test Generation

To improve the contribution coverage of DNNs and verify whether DeepCon has the potential to expose DNN defects, we also develop *DeepCon-Gen*, which can activate new contributions that are not activated. Given a seed input x and an inactivated contribution u = $(con.in, con)$, the target of DeepCon-Gen is to mutate the seed input x to activate the inactivated contribution u. If we can make both the contribution $(con.in, con)$ and the output of neuron $con.out$ larger, it is possible that contribution u can be activated. Thus, we can transform the test generation into a joint optimization problem.

Algorithm 2 shows how DeepCon-Gen works. The joint optimization target obj is constructed by stacking $u(con, x)$ and $n(con.out, x)$, in which, $u(con, x)$ denotes the contribution of $con.in$ to $con.out$, and $n(con.out, x)$ denotes the output of $con.out$ (Line 7). For scalability, DeepCon-Gen uses a gradient ascent-based algorithm (i.e., $gradientAscent$ in Line 8) to iteratively optimize the joint optimization obj so that the input seed x mutates towards the direction that can activate the inactivated contribution u (Line 6−16). In the implementation of $gradientAscent$, we simply adopt a variant of iterative-FGSM [16] to mutate x (and other advanced gradient algorithms such as Adadelta [33] and Adam [34] are also applicable). In order to prevent the hopeless optimization from consistently occupying computing resources, we limit the iterations of optimizations (e.g., maxIter in Line 6). If both contribution u and neuron $con.out$ are activated (Line 13), we successfully generate a test that activates u.

Note that, the gradients of $u(con, x)$ and $n(con.out, x)$ with respect to x are different. This may lead to obj over-optimize towards one of them and ignore the other. To overcome this problem, we adopt a penalty strategy (i.e., adding a soft hinge loss [38]) and make sure $u(con, x)$ and $n(con.out, x)$ can be activated simultaneously. Specifically, we put penalties $pCon$ and $pOut$ on $u(con, x)$ and $n(con.out, x)$, respectively (Line 3−4 and 7). Initially, we set $pCon$ and $pOut$ to a large number, e.g., initMaxVal. If a part, e.g., $n(con.out, x)$ in obj, is activated, we ignore the following changes of x towards the direction of this part (Line 9−12). This strategy can make the obj be optimized towards the direction of the other part and make it be activated quickly.

Algorithm 2: Contribution coverage guided test generation

Input: $seeds$ (Initial test inputs), $coverage$ (Recorded activated contributions)
Output: $tests$

```
1  for x ∈ seeds do
2      u = Randomly select an inactivated contribution (con.in, con)
3      pCon = initMaxVal /*Penalty for the contribution */
4      pOut = initMaxVal /*Penalty for con's output neuron */
5      iter = 0
6      while iter++ < maxIter do
7          obj = min(u(con, x), pCon) + min(n(con.out, x), pOut)
8          x = gradientAscent(▽_x obj, x, ϵ) /* ϵ is the step size */
9          if isAct(con.out, x) and pOut == initMaxVal then
10             pOut = min(n(con.out, x), pOut)
11         if isAct(u, x) and pCon == initMaxVal then
12             pCon = min(u(con, x), pCon)
13         if isAct(con.out, x) ∧ isAct(u, x) then
14             tests.put(x)
15             update coverage
16             break
17     if desired coverage achieved then
18         return tests
```

V. Experiments

We implement DeepCon and DeepCon-Gen using Keras 2.2.4 [26] with backend TensorFlow 1.14.0 [25]. All our experiments are run on servers with CentOS 7.6. The hardware of each server is a 20-core 2.20GHz Xeon CPU with 128G RAM and 4 NVIDIA TITAN V GPU with 12G Video memory.

A. Experimental Setup

We first select three classic LeNet series models (i.e., LeNet-1, LeNet-4 and LeNet-5 [8]) on MNIST dataset [20] to evaluate DeepCon and DeepCon-Gen. As the scalability is one of the most important factors for a coverage criterion [11], we further select two much larger pre-trained DNN models VGG-19 [18] and ResNet-50 [19] on ImageNet dataset [21] to evaluate DeepCon and DeepCon-Gen. Note that both VGG-19 and ResNet-50 models have achieved competitive records in the competition of ILSVRC [21], [39].

MNIST is a large-scale handwritten digit recognition database, containing 60,000 training examples and 10,000 test examples. Each example is a 28 * 28 pixel single-channel grayscale image, and its category label is any number from 0 to 9. For MNIST dataset, we use the re-trained LeNet-1,

TABLE I
DNNS AND DATASETS USED IN DEEPCON AND DEEPCON-GEN.

Dataset	Description	DNN	# Neurons	# Cons
MNIST	Digit recog.	LeNet-1	619	5,932
		LeNet-4	907	66,798
		LeNet-5	1,027	105,102
ImageNet	General images	VGG-19	40,747	125,857,984
		ResNet-50	50,283	15,433,920

LeNet-4, and LeNet-5 models as our evaluation subject. Note that LeNet-1, LeNet-4, and LeNet-5 contain 7, 8, and 9 layers, respectively, and their network architectures are very similar.

ImageNet is a super large-scale general image recognition database with 1,000 categories. It contains more than 1.4 million training examples and 100,000 test examples and is one of the most popular datasets used in various computer vision competitions [21]. Each ImageNet example is a 224*224*3 pixel image. For ImageNet, we select two state-of-art pre-trained DNNs VGG-19 and ResNet-50 as our evaluation subject. The architectures of VGG-19 with 26 layers and ResNet-50 with 177 layers are much more complex than those of LeNet models.

For LeNet models on MNIST, we perform the evaluation on the whole test set, i.e., 10,000 test examples. For VGG-19 and ResNet-50 on ImageNet, considering the simplicity and effectiveness of experiments, we perform the evaluation on 5000 randomly sampled examples from the test set. The details of our adopted DNNs and data sets are shown in Table I.

We evaluate DeepCon and DeepCon-Gen by investigating the following three research questions.

RQ1: *How strong is the coverage of DeepCon? Can it scale well to large DNNs?*

RQ2: *Compared with existing works, how good does DeepCon perform in terms of coverage strength and adversarial example detection?*

RQ3: *How effectively can DeepCon-Gen activate inactivated contributions and find unexpected behaviors of DNNs?*

We first evaluate the coverage strength of DeepCon to verify whether it can identify much more different tests. Since a good coverage criterion should also have a good (potential) usefulness in other related tasks such as adversarial example detection [10]–[12], [35] , we directly compare DeepCon with existing works in terms of coverage strength and adversarial example detection. Lastly, we evaluate DeepCon-Gen on the remaining inactivated contributions to verify whether it can effectively activate these contributions and find defects.

B. Coverage Results

In this subsection, we evaluate the coverage strength of contribution coverage criterion on entire DNNs and each layer of DNNs.

1) Contribution Coverage of DNNs: Figure 3 shows the growth trend of contribution coverage (ConC) on increasing test inputs. The experimental results show that: (1) Under the weakest setting of $t > 0$, the ConC of each DNN has not reached 100% after all test inputs are fed. The ConC of each DNN grows as the test input increases, and the growth trend becomes slower and slower. This means that more different test inputs can activate more contributions, and the remaining contributions are more and more difficult to be activated. (2) A larger t will result in a lower ConC, and as the value of t becomes larger, the corresponding ConC drops much faster. For example, for all LeNet models and VGG-19, the growth trend lines of ConC with $t > 0$ and $t > 0.25$ are almost overlapped, while the growth trend lines of ConC with $t > 0.5$ is much higher than that with $t > 0.75$. The experimental results mean that during the prediction-making of a DNN on a test input, only a few neurons with large contributions can influence or even determine the prediction result. (3) DeepCon can scale well to large DNNs such as VGG-19 and ResNet-50. This is because the contribution computation cost of a DNN on a given input is approximately equal to the prediction cost of the DNN on the input, and the corresponding extraction cost of activated contributions is linear.

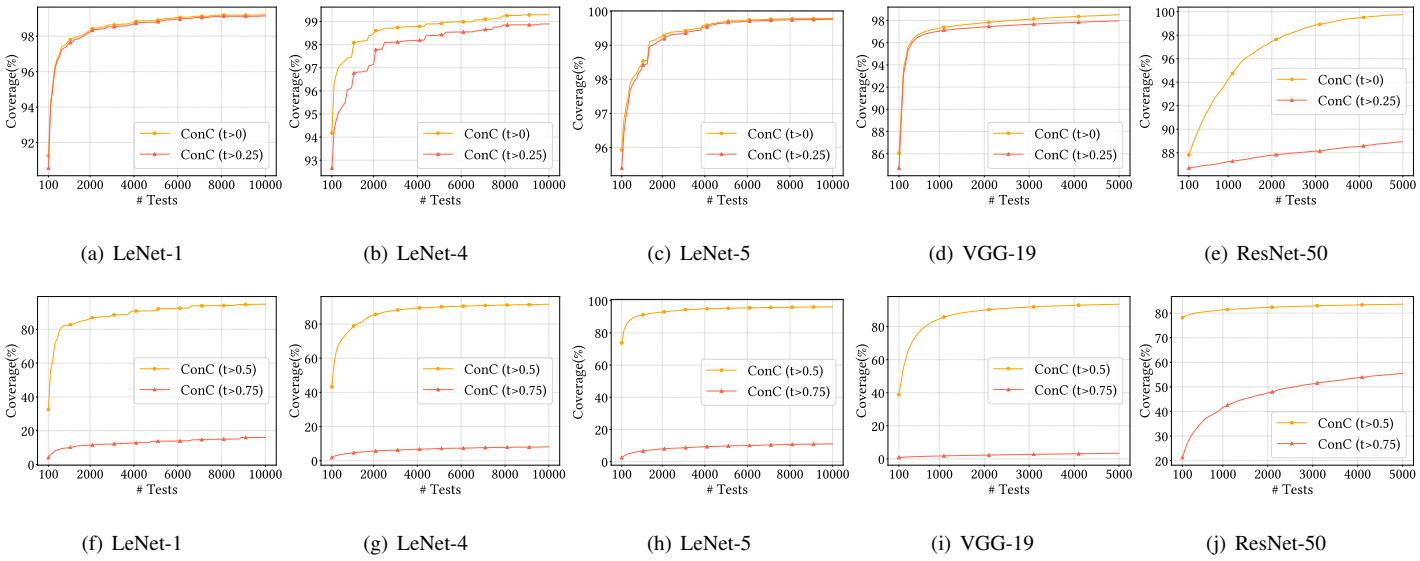
Answer to RQ1. Under the setting of $t > 0$, the contribution coverage of any of these DNNs grows with the increase of test inputs and does not reach 100% even on all test inputs. Moreover, contribution coverage criterion can scale well to large DNNs.

2) Contribution Coverage of Each Layer: DNN extracts features layer-by-layer. The testing adequacy of each layer in DNN can be useful for DNN researchers and developers. Different layers usually have different numbers of neurons and connections, the layer with more neurons and connections may has a greater influence on the result of ConC of the entire DNN than that with fewer neurons and connections. To better reflect the testing adequacy of DNNs, we evaluate the ConC of each layer of DNNs.

The experimental results of ConC_is are shown in Figure 4. First, similar to ConC, a larger setting of t leads to a lower value of ConC_i. Second, the ConC_i becomes lower as the layer is deeper, even though this decreasing trend is not strictly monotonically decreasing. Third, compared with Figure 3, when ConC is very high, some ConC_is are still very lower. For example, as shown in Figure 3(i) and Figure 4(d), when the ConC (with $t > 0.5$) of VGG-19 on 5000 test inputs is 93.54%, the corresponding ConC_{24} and ConC_{25} is only 75.56% and 6.01% respectively. Thus, ConC_i may be be able to evaluate the testing adequacy of DNNs in a more fine-grained manner.

C. Comparison with Existing Works

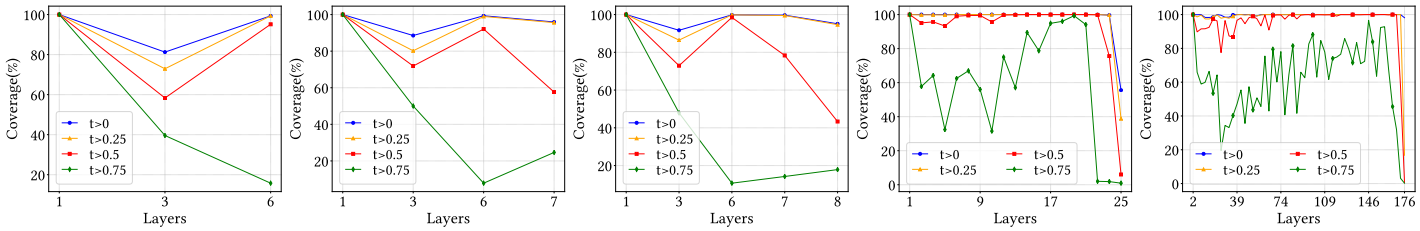
Researchers often use the coverage strength of a coverage criterion to reflect how many different test inputs it can identify. Thus, we first compare the coverage strength of contribution coverage with another coverage criterion that can be fairly compared. Furthermore, a good coverage criterion should have a potential usefulness in other closely related tasks such as adversarial example detection [11], [12]. We then con-



(a) LeNet-1 (b) LeNet-4 (c) LeNet-5 (d) VGG-19 (e) ResNet-50

(f) LeNet-1 (g) LeNet-4 (h) LeNet-5 (i) VGG-19 (j) ResNet-50

Fig. 3. The growth trend of contribution coverage.



(a) ConC_i of LeNet-1 (b) ConC_i of LeNet-4 (c) ConC_i of LeNet-5 (d) ConC_i of VGG-19 (e) ConC_i of ResNet-50

Fig. 4. Contribution coverage of each layers.

duct a fair comparison between DeepCon and existing works to verify whether DeepCon can achieve higher performance in adversarial example detection.

1) Coverage Strength: As we analyzed in Section II-B and Section III, the activated contribution extraction method in DeepCon is completely different from the activated neuron extraction method in NOB coverage criteria. The meaning of the hyper-parameter t in contribution coverage and NOB coverage criteria is dramatically different. Therefore, we cannot directly compare them even under the same value of t. The contribution defined in DeepCon is the combination of a neuron and the connection weight it connects to the next-level neuron, thus the neuron coverage in DeepXplore (DNC) may be a potential object for fair comparison.

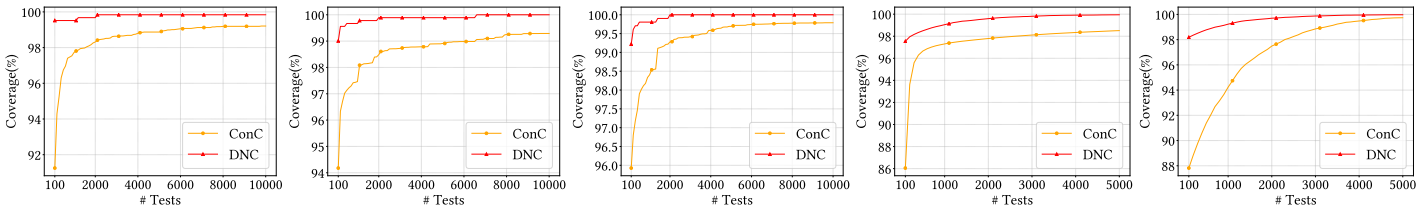
A fair setting for the comparison between ConC and DNC should satisfy that the number of input neurons of activated contributions in ConC should be approximately equal to the number of activated neurons in DNC. This means that only when we consider the non-zero contributions in ConC as activated contributions and view the neurons with non-zero outputs in DNC as activated neurons, the comparison of these two coverage criteria can be relatively fair.

The comparison results are shown in Figure 5. Overall, the coverage strength of ConC is stronger than that of DNC, and the change of ConC with increasing test inputs is more obvious than that of DNC. For example, the DNC of ResNet-50 on 100 test inputs reaches about 98%, while the ConC reaches lower than 88%. The ConC of Resnet-50 can identify almost all 5000 test inputs.

The reason why ConC is stronger than DNC is that ConC fully considers the neuron and the connection weights it connects to next-level neurons. Take the fully connected DNN as an example. When we neglect the differences caused by their extraction methods, a neuron $n_{i\text{-}1,h}$ will correspond to s_i contributions, i.e., $|\{(n_{i\text{-}1,h}, c^i_{h,j})\}| = s_i$, where $0 \leqslant j < s_i$. Thus, under this assumption, the coverage strength of ConC will be s_i times that of DNC.

2) Adversarial Example Detection: A good coverage criterion should also have good potential usefulness in defect detection [12]. Existing works [10]–[12], [35] all use adversarial examples to help verify the potential usefulness of their coverage criteria. In this subsection, we directly conduct an adversarial example detection task to compare the usefulness between DeepCon and NOB coverage criteria.

As demonstrated in DeepXplore that test inputs from the same category tend to share overlapped neurons. We also get



(a) LeNet-1 (b) LeNet-4 (c) LeNet-5 (d) VGG-19 (e) ResNet-50

Fig. 5. Comparison between ConC and DNC.

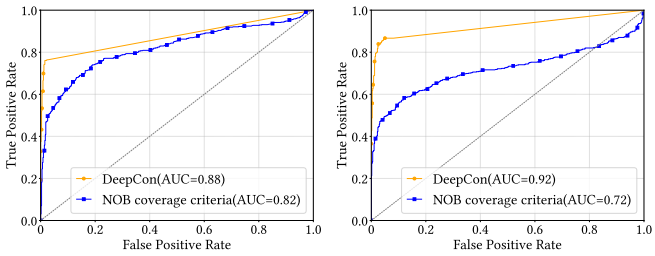
a similar observation in DeepCon, normal inputs from the same category tend to cover some of the same neurons, while adversarial examples tend to cover different neurons. Based on this observation, we employ DeepCon and NOB coverage criteria to detect adversarial examples. The adversarial example detection method is described as follows.

Firstly, we use DeepCon to extract activated contributions $\{\widetilde{U}^i(x)|0 < i \leqslant l\text{-}1\}$ of DNN on each correctly predicted example from training set, and aggregate the activated contributions of each example from the same category to obtain *category-contributions* $\{\widetilde{U_c}^i|0 < i \leqslant l\text{-}1\}$. we can further obtain *category-connections* $\{\widetilde{C_c}^i|0 < i \leqslant l\text{-}1\}$ from $\{\widetilde{U_c}^i|0 < i \leqslant l\text{-}1\}$. Note that, here, the *category-connections* represent the overlapped adjacent neuron pair set activated by training examples from the same category. Secondly, for a test input x, if its activated adjacent neuron pairs $\{\widetilde{C}^i(x)|0 < i \leqslant l\text{-}1\}$ satisfies $\frac{\{\widetilde{C}^i(x)\}\cap\{\widetilde{C_c}^i\}}{\{\widetilde{C}^i(x)\}} < \gamma$, we consider x is an adversarial example.

Similar to the above two steps, we also use NOB coverage criteria to perform adversarial example detection. The difference is that we use NOB coverage criteria to extract the same number of activated neurons as that in $\{\widetilde{U}^i(x)\}$ and add each activated adjacent neuron pair to *category-connections*.

We use targeted-FGSM [40] to generate adversarial examples, and use DeepCon and NOB coverage criteria to perform adversarial example detection, respectively. Considering the high cost of crafting adversarial examples, we only perform adversarial example detection on LeNet-4 and LeNet-5 over MNIST. The experimental results are shown in Figure 6. On average, the Area Under the ROC Curve (AUC) [22] of DeepCon exceeds that NOB coverage criteria by 13.00% (the larger the AUC, the better the adversarial example detection). Specifically, when DeepCon achieves the AUC of 0.88 and 0.92 on LeNet-4 and LeNet-5, respectively, NOB coverage criteria only achieves 0.82 and 0.72, respectively.

The in-depth reason behind the experiments may be that DeepCon fully considers the importance of the output of a neuron and the connection weights it connects to next-level neurons and can extract more *true* decision-logics of DNNs, while NOB coverage criteria only considers the output of a neuron and ignores the connection weights emitted by the neuron.



(a) LeNet-4 (b) LeNet-5

Fig. 6. The comparison in adversarial example detection.

Answer to RQ2. From the perspective of coverage strength, since one neuron corresponds to multiple contributions, the coverage strength of contribution coverage criterion is stronger than that of DNC. From the perspective of usefulness in the adversarial example detection, DeepCon can achieve significantly better performance than NOB coverage criteria.

D. Results of DeepCon-Gen

We evaluate DeepCon-Gen to verify whether it can activate inactivated contributions. It is known that adversarial examples are pervasive and often regarded as DNN defects. Following previous works [10], [12], [35], we also utilize the proportion of adversarial examples in the generated examples to investigate whether contribution coverage criterion has the potential usefulness of exposing DNN defects. Note that, the main purpose of DeepCon-Gen does not include crafting adversarial examples, but utilizing the generated adversarial examples as DNN defects to prove that contribution coverage has the potential to expose defects.

Based on the experiments of Section V-B1, we try to activate the remaining inactivated contributions of all models except VGG-19. There are a large number of inactivated contributions in VGG-19. For simplicity, we only try to activate 100,000 contributions with larger connection weights in the deepest few layers of VGG-19. (1) These remaining inactivated contributions are more difficult to activate than those contributions that have been activated by original test inputs. (2) The number of these inactivated contributions is limited and they can be activated in parallel. For easy comparison, we also perform DNC guided test generation. The experimental results are

shown in Table II, in which *un-cs/ns* represents inactivated contributions / neurons, *Alg2. cs/ns* represents the activated contributions / neurons by Algorithm 2, and *Advs* represents adversarial examples.

First, DeepCon-Gen can effectively activate the remaining inactivated contributions. Except for LeNet1, DeepCon-Gen can activate more than 95% of the remaining inactivated contributions. Second, when the DNC fails to generate any adversarial examples, DeepCon-Gen can still generate a large number of adversarial examples, which means that DeepCon can bring more chances to expose defects than DNC. For LeNet-1, LeNet-4 and LeNet-5, 84.21%, 11.44% and 20.28% of the tests generated by DeepCon-Gen are adversarial examples, respectively. For VGG-19 and ResNet-50, 84.21% and 99.84% of the tests generated by DeepCon-Gen are adversarial examples, respectively. The reason for the high proportion of adversarial examples in the generated tests for VGG-19 and ResNet-50 is that almost all of these inactivated contributions come from the deepest few layers.

We also evaluate DeepCon-Gen with other settings of t. When $t > 0.25$, the experimental results are very close to the results of $t > 0$. However, when $t > 0.5$ and $t > 0.75$, DeepCon-Gen can only activate 20%-30% and less than 10% of the remaining contributions of LeNet models and VGG-19, respectively. This can be explained as follows. Each hidden layer of LeNet models and VGG-19 is followed by a ReLu function, which makes the output of these layers non-negative. When some connection weights between these layers are negative, their corresponding contributions will hardly be greater than a given larger t. Therefore, the setting of including ReLu in each hidden layer may restrict the representation capacity or expression power of DNNs. This may also be why ResNet-v1 [19] directly abandons the use of ReLu in each shortcut connection branch, and why ResNet-v2 [41] tends to place ReLu before the linear layer in the residual block and remove it between two sequential residual blocks.

Answer to RQ3. On average, DeepCon-Gen can effectively activate 93.63% of the remaining inactivated contributions, and 62.60% of the generated tests can lead to the mis-predictions of DNNs. The high proportion of adversarial examples in the generated examples indicates that contribution coverage has higher potential usefulness in exposing DNN defects.

VI. Discussion

In this section, we discuss the potential threats and limitations of our work.

Threats to Validity. To comprehensively and thoroughly evaluate DeepCon and DeepCon-Gen, we select five different-sized DNNs and two publicly accessible large-scale datasets. For the large-sized DNNs VGG-19 and ResNet-50 on ImageNet dataset, we randomly sampled 5,000 test inputs to evaluate the coverage strength of contribution coverage criterion. Since the selected test inputs are not the whole ImageNet test set, our experimental results may not fully reflect the testing adequacy of VGG-19 and ResNet-50 on ImageNet. Besides, some advanced models mainly used in the text field, such as Transformer [42] and BERT [43], are not be investigated in the experiments. Fortunately, the experimental subjects we used are enough to prove that DeepCon can not only completely cover neurons and connection weights of DNNs, but also can scale well to large-size DNN models.

TABLE II
Testing Results of DeepCon-Gen on Different DNNs

Coverage	DNN	# un-cs/ns	% Alg2. cs/ns	% Advs
ConC	LeNet-1	47	80.85%	84.21%
	LeNet-4	474	97.68%	11.44%
	LeNet-5	223	97.31%	20.28%
	VGG-19	1,875,097	96.47%	99.88%
	ResNet-50	40,579	95.88%	97.24%
DNC	LeNet-1	1	100.00%	0.00%
	LeNet-4	0	0.00%	0.00%
	LeNet-5	0	0.00%	0.00%
	VGG-19	22	40.90%	0.00%
	ResNet-50	14	57.14%	0.00%

In the comparison with the existing work, we only chose the fundamental coverage criterion DNC. This is because the extraction method of activated neurons of DeepCon is dramatically different from NOB coverage criteria, which makes the comparison with other advanced NOB coverage criteria inconvenient and unfair. As compensation, we potentially compared DeepCon with paired-neurons coverage in the experiments of Section V-C2. The experimental results show that DeepCon has a better performance in detecting adversarial examples.

Limitations. In DeepCon, we use contribution coverage to reflect the internal logic coverage of DNNs, which makes DeepCon can thoroughly cover both the outputs of neurons and the connection weights they emit. However, similar to NOB coverage criteria [10]–[12], [35], contribution coverage cannot be directly generalized to stateful DNNs, such as RNN and LSTM [44], either. Fortunately, we can expand the calculation process of a stateful DNN into a stateless sequence, and then use contribution coverage criterion to measure the testing adequacy of DNNs.

VII. Related Work

A. Testing of DNNs

Traditionally, developers and researchers often use the accuracy of a DNN on the test set to reflect the quality of the DNN. This black box testing method may not be able to effectively detect various unexpected predictions of DNNs [11].

To thoroughly test DNNs, Pei et al. [10] first proposed a neuron coverage criterion to reflect the internal logic coverage of DNNs. They take a neuron as the basic internal logic of DNN and use the output of the neuron on a test input to determine its activation state. If the output of a neuron is greater than a given threshold, the neuron is considered as activated. DeepTest [45] proposed a gray-box approach to use

neuron coverage to guide test generation. They performed a finite number of affine transformations (eg., rotation, translation, scaling) on seeds to systematically generate tests. The generated tests successfully expose various erroneous behaviors of autonomous driving systems.

Ma et al. [11] argued that the range of output values of neurons in different layers may vary largely, and it is inappropriate to set the same threshold for these neurons to indicate their activation states. They proposed a k-multisection neuron coverage (KMNC) to overcome this problem. Besides, they also proposed other two neuron-level and two layer-level coverage criteria. DeepHunter [46] proposed a fuzz testing framework to generate sematic-preserved tests to hunt the defects of DNNs. They use the coverage criteria proposed in [10], [11] as feedback and perform mutations (a limited number of affine transformations) on input seeds to craft tests. Similarly, Odena et al. [47] proposed Tensorfuzz to debug DNNs with coverage-guided fuzzing.

DeepXplore [10] and DeepGauge [11] ignored the influence between neurons. To test the independent influence between two adjacent neurons / features, Sun et al. [12] proposed a set of adapted MC/DC coverage criteria for DNNs. Their experiments on small DNNs show that their coverage criteria have higher bug detection capabilities than random testing. However, their covering method needs to freeze the outputs of all other neurons in the same layers with the adjacent neurons under test. This makes their coverage criteria cannot easily scale to large DNNs [11], [13]. With the guidance of coverage criteria proposed in [10]–[12], Sun et al. [48] proposed DeepConcolic to perform concolic testing for DNNs. Due to the limitation of the constraint solver, this work can hardly scale well to large DNNs either.

Inspired by traditional combinatorial testing, Ma et al. [35] proposed a set of t-way (t neurons in the same layer) combination coverage criteria to detect the local robustness of DNNs. Their experiments on two small DNNs shown that DeepCT is effective in testing DNNs. However, due to the huge combination space of different neurons, whether their coverage criteria can scale well to large and complex DNNs also remains to be verified in practice.

B. Attack and Defense of DNNs

The accuracy of a DNN on the test set has been the most important or even the only indicator to evaluate the quality of DNNs. However, adversarial attacks against DNNs in recent years shown that DNNs with high accuracy can be easily broken by some attack algorithms. The robustness of DNNs is seriously threatened. Adversarial attack refers to adding some carefully designed slight perturbations to the original example, causing the DNN to make misprediction with high confidence.

Szegedy et al. [49] first turned on DNN's adversarial attack and defense. They employed the L-BFGS method to solve the attack objective to craft adversarial examples. Goodfellow et al. [40] proposed the hypothesis that the features of the DNN high-dimensional feature space are linear, and proposed an FGSM attack method to generate adversarial examples. Following this research direction, more attacks methods such as JSMA [50], BIM&ILCM [16], One-pixel [51], C&W [38] and DeepFool [52]–[55], were proposed.

The research of adversarial attack further promoted the research of adversarial defense. The adversarial defense can be classified into proactive defense and passive defense [56]–[58]. For more information about adversarial defense, we recommend readers to read [1], [59].

VIII. Conclusion

Deep learning systems have been widely adopted in various safety-critical domains, which put forward higher requirements on the robustness of DL systems. Existing coverage criteria in DL systems all use the output of a neuron to determine the activation state of the neuron and ignore the connection weights it emits.

In this paper, we propose DeepCon, a novel contribution coverage criterion. DeepCon can simultaneously cover the output of a neuron and the connection weights it connects to other neurons and can scale well to large-sized DNNs. The experiments on different DNNs and datasets show that DeepCon can well present the testing adequacy of DNNs and has higher usefulness in other application tasks such as adversarial example detection. We further propose DeepCon-Gen, which can effectively activate the inactivated contributions, and most of the generated tests can lead to mispredictions.

In the future, we plan to investigate advanced coverage criteria based on contribution coverage, e.g., paired contribution coverage criterion. We also plan to explore more complex contribution coverage-based defense methodologies for adversarial example detection and input validation.

Acknowledgment

This work was partially supported by National Key R&D Program of China (2018YFC0831202), National Natural Science Foundation of China (61802381, 61732019), Frontier Science Project of Chinese Academy of Sciences (QYZDJ-SSW-JSC036), and Youth Innovation Promotion Association at Chinese Academy of Sciences. Both Wensheng Dou and Jie Liu are the corresponding authors of this paper.

References

[1] N. Akhtar and A. Mian, "Threat of adversarial attacks on deep learning in computer vision: A survey," *IEEE Access*, vol. 6, pp. 14 410–14 430, 2018.

[2] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012, pp. 3642–3649.

[3] G. Hinton, L. Deng, D. Yu, G. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, B. Kingsbury *et al.*, "Deep neural networks for acoustic modeling in speech recognition," *IEEE Signal Processing Magazine*, vol. 29, 2012.

[4] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *IEEE Computational intelligenCe magazine*, vol. 13, no. 3, pp. 55–75, 2018.

[5] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[6] C. Ziegler, "A google self-driving car caused a crash for the first time," *The Verge*, 2016.

[7] F. Lambert, "Understanding the fatal Tesla accident on autopilot and the NHTSA probe," *Electrek*, vol. 1, 2016.

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

[9] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[10] K. Pei, Y. Cao, J. Yang, and S. Jana, "DeepXplore: Automated whitebox testing of deep learning systems," in *Proceedings of USENIX Symposium on Operating Systems Principles (SOSP)*, 2017, pp. 1–18.

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

[12] Y. Sun, X. Huang, D. Kroening, J. Sharp, M. Hill, and R. Ashmore, "Testing deep neural networks," *arXiv preprint arXiv:1803.04792*, 2018.

[13] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Transactions on Software Engineering*, 2020.

[14] N. Carlini and D. Wagner, "Adversarial examples are not easily detected: Bypassing ten detection methods," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. ACM, 2017, pp. 3–14.

[15] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," *arXiv preprint arXiv:1704.01155*, 2017.

[16] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," *arXiv preprint arXiv:1607.02533*, 2016.

[17] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[18] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[20] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[21] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "ImageNet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[22] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 233–240.

[23] N. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, pp. 1–12.

[24] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda," in *ACM SIGGRAPH 2008 Classes*, 2008.

[25] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *Proceedings of USENIX Symposium on Operating Systems Design and Implementation*, 2016, pp. 265–283.

[26] F. Chollet *et al.*, "keras. github repository," *https://github.com/fchollet/keras. Accessed on*, vol. 25, p. 2017, 2015.

[27] N. Ketkar, "Introduction to pytorch," in *Deep learning with python*. Springer, 2017, pp. 195–208.

[28] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 315–323.

[29] P. F. Brown, P. V. Desouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai, "Class-based n-gram models of natural language," *Computational linguistics*, vol. 18, no. 4, pp. 467–479, 1992.

[30] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *ICLR*, 2013.

[31] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[32] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International conference on machine learning*, 2013, pp. 1139–1147.

[33] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[34] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[35] L. Ma, F. Juefei-Xu, M. Xue, B. Li, L. Li, Y. Liu, and J. Zhao, "DeepCT: Tomographic combinatorial testing for deep learning systems," in *Proceedings of IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2019, pp. 614–618.

[36] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Proceedings of European Conference on Computer Vision*, 2014, pp. 818–833.

[37] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, "Understanding neural networks through deep visualization," *arXiv preprint arXiv:1506.06579*, 2015.

[38] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Proceedings of IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 39–57.

[39] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *arXiv: Computer Vision and Pattern Recognition*, 2014.

[40] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[41] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European conference on computer vision*. Springer, 2016, pp. 630–645.

[42] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[43] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[44] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[45] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of International Conference on Software Engineering*, 2018, pp. 303–314.

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

[47] A. Odena, C. Olsson, D. Andersen, and I. Goodfellow, "Tensorfuzz: Debugging neural networks with coverage-guided fuzzing," in *International Conference on Machine Learning*. PMLR, 2019, pp. 4901–4911.

[48] Y. Sun, X. Huang, D. Kroening, J. Sharp, M. Hill, and R. Ashmore, "Deepconcolic: testing and debugging deep neural networks," in *Proceedings of International Conference on Software Engineering*, 2019, pp. 111–114.

[49] C. Szegedy, W. Zaremba, I. Sutskever, J. B. Estrach, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *2nd International Conference on Learning Representations, ICLR 2014*, 2014.

[50] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Proceedings of IEEE European Symposium on Security and Privacy (EuroS&P)*, 2016, pp. 372–387.

[51] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks," *IEEE Transactions on Evolutionary Computation*, 2019.

[52] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2574–2582.

[53] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1765–1773.

[54] S. Sarkar, A. Bansal, U. Mahbub, and R. Chellappa, "Upset and angri: Breaking high performance image classifiers," *arXiv preprint arXiv:1707.01159*, 2017.

[55] M. Cisse, Y. Adi, N. Neverova, and J. Keshet, "Houdini: Fooling deep structured prediction models," *arXiv preprint arXiv:1707.05373*, 2017.

[56] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *Proceedings of IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 582–597.

[57] S. Gu, P. Yi, T. Zhu, Y. Yao, and W. Wang, "Detecting adversarial examples in deep neural networks using normalizing filters," *UMBC Student Collection*, 2019.

[58] C. Xie, J. Wang, Z. Zhang, Z. Ren, and A. Yuille, "Mitigating adversarial effects through randomization," in *International Conference on Learning Representations*, 2018.

[59] H. Xu, Y. Ma, H. C. Liu, D. Deb, H. Liu, J. L. Tang, and A. K. Jain, "Adversarial attacks and defenses in images, graphs and text: A review," *International Journal of Automation and Computing*, vol. 17, no. 2, pp. 151–178, 2020.