DeepGauge: Multi-Granularity Testing Criteria for Deep Learning Systems

Lei Ma[1,3], Felix Juefei-Xu[2], Fuyuan Zhang[3], Jiyuan Sun[4], Minhui Xue[3], Bo Li[5]
Chunyang Chen[6], Ting Su[3], Li Li[6], Yang Liu[3], Jianjun Zhao[4], and Yadong Wang[1]
[1]Harbin Institute of Technology, China [2]Carnegie Mellon University, USA [3]Nanyang Technological University, Singapore
[4]Kyushu University, Japan [5]University of Illinois at Urbana–Champaign, USA [6]Monash University, Australia
Contact author: malei@hit.edu.cn.

ABSTRACT

Deep learning (DL) defines a new data-driven programming paradigm that constructs the internal system logic of a crafted neuron network through a set of training data. We have seen wide adoption of DL in many safety-critical scenarios. However, a plethora of studies have shown that the state-of-the-art DL systems suffer from various vulnerabilities which can lead to severe consequences when applied to real-world applications. Currently, the testing adequacy of a DL system is usually measured by the accuracy of test data. Considering the limitation of accessible high quality test data, good accuracy performance on test data can hardly provide confidence to the testing adequacy and generality of DL systems. Unlike traditional software systems that have clear and controllable logic and functionality, the lack of interpretability in a DL system makes system analysis and defect detection difficult, which could potentially hinder its real-world deployment. In this paper, we propose *DeepGauge*, a set of multi-granularity testing criteria for DL systems, which aims at rendering a multi-faceted portrayal of the testbed. The in-depth evaluation of our proposed testing criteria is demonstrated on two well-known datasets, five DL systems, and with four state-of-the-art adversarial attack techniques against DL. The potential usefulness of *DeepGauge* sheds light on the construction of more generic and robust DL systems.

KEYWORDS

Deep learning, Neural networks, Testing criteria, White-box testing, Software testing

1 INTRODUCTION

Deep learning (DL) systems have gained great popularity in various applications, *e.g.*, speech processing [26], medical diagnostics [12], image processing [11], and robotics [58]. A deep neural network (DNN), as a type of deep learning systems, is the key driving force behind recent success. However, DNN-based software systems, such as autonomous driving, often exhibit erroneous behaviors that lead to fatal consequences. For example, several accidents [21] have been reported due to autonomous vehicle's failure to handle unexpected/corner-case driving conditions.

One of the trending research areas is to investigate the cause of vulnerability in DL systems by means of generating adversarial test examples for image- and video-based DL systems. Such carefully learned pixel-level perturbations, imperceptible to human eyes, can cause the DL-based classification system to output completely wrong decisions with high confidence [20]. Ever since the inception of adversarial attacks on the DL systems, more and more research has been dedicated to building up strong attackers [6, 25, 55, 60]. As a consequence, better defense mechanisms in DL systems against adversarial attacks are in dire need. Various techniques to nullify adversarial attacks and to train a more robust DL system are emerging in recent studies [18, 23, 41, 43, 45, 51, 56]. Together, research in both realms forms a virtuous circle and blazes a trail for our better understanding of how to build more generic and robust DL systems.

However, what is still lacking is a systematic way of gauging the testing adequacy of given DL systems. Current studies focus only on pursuing high accuracy of DL systems as a testing criterion, for which we show several caveats as follows. ***First***, measuring the software quality from DL output alone is superficial in the sense that fundamental understanding of the DL internal neuron activities and network behaviors is not touched upon. We agree that it could be an indicator of DL system quality and generality, but it is far from complete, and oftentimes unreliable. ***Second***, a criterion solely based on DL output will rely heavily on how representative the test data are. Having achieved high-performance DL output does not necessarily mean that the system is utmost generic, and achieving low-performance does not indicate the opposite either. A DL model can be immune to many known types of adversarial attacks, but may fail from unseen attacks. This is because such a criterion based only on DL outputs is far from being comprehensive, and it leaves high risks for currently cocooned DL systems to be deployed in the real-world environment where newly evolved adversarial attacks are inescapable. ***Third***, any DL system that passes systematic testing should be able to withstand all types of adversarial attacks to some extent. Such generality upon various attacks is of vital importance for DL systems to be deployed. But apparently this is not the case, unless we stick to a set of more comprehensive gauging criteria. We understand that even the most comprehensive gauging criteria would not be able to entirely eliminate risks from adversarial attacks. Nevertheless, by enforcing a suitable set of testing criteria, we hope that a DL system could be better tested to facilitate the construction of a more generic and robust deep learning system.

Towards addressing the aforementioned limitations, *a set of* testing criteria is needed, as opposed to the sole criterion based on DL decision output. In addition to being scalable, the proposed criteria will have to monitor and gauge the neuron activities and intrinsic network connectivity at various granularity levels, so that a multi-faceted in-depth portrayal of the DL system and testing quality measures become desirable.

In this work, we are probing this problem from a software engineering and software testing perspective. At a high level, erroneous behaviors appeared in DNNs are analogous to logic bugs in traditional software. However, these two types of software are

fundamentally different in their designs. Traditional software represents its logic as control flows crafted by human knowledge, while a DNN characterizes its behaviors by the weights of neuron edges and the nonlinear activation functions (determined by the training data). Therefore, detecting erroneous behaviors in DNNs is different from detecting those in traditional software in nature, which necessitates novel test generation approaches.

To achieve this goal, the very first step is to precisely define a set of suitable coverage criteria, which can guide test design and evaluate test quality. Despite a number of criteria existing for traditional software, *e.g.*, statement, branch, data-flow coverage, they completely lose effect in testing DNNs. To the best of our knowledge, the design of testing coverage criteria for DNNs is still at the early stage [38, 47]. Without a comprehensive set of criteria, (1) designing tests to cover different learned logics and rules of DNNs is difficult to achieve. Consequently, erroneous behaviors may be missed; (2) evaluating test quality is biased, and the confidence of obtained testing results may be overestimated. In this paper, we propose *DeepGauge*—a set of testing criteria based on multi-level and -granularity coverage for testing DNNs and measure the testing quality. Our contributions are summarized as follows:

- Our proposed criteria facilitate the understanding of DNNs as well as the test data quality from different levels and angles. In general, we find defects could potentially distribute on both major function regions as well as the corner-case regions of DNNs. Given a set of inputs, our criteria could measure to what extent it covers the main functionality and the corner cases of the neurons, where DL defects could incur. Our evaluation results reveal that the existing test data of a given DL in general skew more towards testing the major function region, with relatively few cases covering the corner-case region.
- In line with existing test data of DNNs, we evaluate the usefulness of our coverage criteria as indicators to quantify defect detection ability of test data on DNNs, through generating new adversarial test data using 4 well-known adversarial data generation algorithms (*i.e.*, Fast Gradient Sign Method (FGSM) [20], Basic Iterative Method (BIM) [31], Jacobian-based Saliency Map Attack (JSMA) [37] and Carlini/Wagner attack (CW) [8]). The extensive evaluation shows that our criteria can effectively capture the difference between the original test data and adversarial examples, where DNNs could and could not correctly recognize, respectively, demonstrating that a higher coverage of our criteria potentially indicates a higher chance to detect the DNN's defects.
- The various criteria proposed behave differently on DNNs *w.r.t.* network complexity and dataset under analysis. Altogether, these criteria can potentially help us gain insights of testing DNNs. By providing these insights, we hope that both software engineering and machine learning communities can benefit from applying new criteria for gauging the testing quality of the DNNs to gain confidence towards constructing generic and robust DL systems.

To the best of our knowledge, this is among the earliest studies to propose multi-granularity testing criteria for deep learning systems, which are mirrored by the test coverage in traditional software testing.

2 PRELIMINARIES

In this section, we first introduce traditional software and then deep learning systems of which the architectural features appearing to be a step above current traditional software. We will see that deep learning fundamentally changes the software development paradigm. Precisely, we try to analogize that the programming language logic execution to traditional software is what the connectivity strength (weights) to a DNN. As we will see below, we are attempting to connect these two counterparts as well as discussing the differences.

2.1 Coverage Criteria in Traditional Software Testing

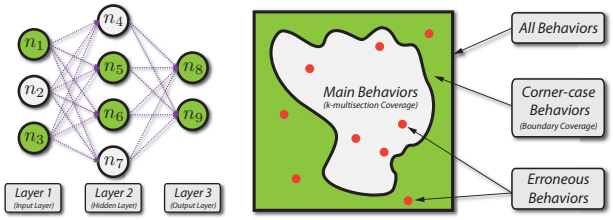
We regard traditional software as any program written in high-level programming languages (*e.g.*, `C/C++`, `Java`, `Python`). Specially, each statement in traditional program performs some certain operation that either transforms the outputs from the previous statement to the next one or changes the program states (*e.g.*, assign new values to variables). *Software defects* (*bugs*) can be introduced by developers due to incorrect implementation, which may cause unexpected outputs or even fail-stop errors (*e.g.*, program crashes).

To detect defects, software testing is one of the most widely adopted software validation techniques in software industry— Given a set of test data, it feeds these test data as inputs to program and validates the correctness of the program's run-time behavior by comparing the actual outputs with expected ones (test oracles); and measures test adequacy by using *coverage criteria*, the important, practical measures to quantify the degree to which the software is tested [36]. The program with higher test coverage often suggests that it has a lower chance of containing defects. Many software testing standards require a software product to be thoroughly tested with high test coverage before shipment, which is used as an indicator and confidence of the software quality. On some safety critical systems, the requirement of some form of test coverage is even 100%. For example, ECSS-E-ST-40C [15] standards demand 100% statement coverage of the software under testing for two critical levels.

In the literature, for traditional software, a number of coverage criteria have already been defined at different levels, to analyze the software run-time behavior from different perspectives, *i.e.*, code level (*e.g.*, statement, branch, data-flow coverage and mutation testing [27, 46, 62]) or model-level (*e.g.*, state and transition coverage [2, 14]) to cater for different testing methods and granularities. Some commonly used test coverage criteria are listed as follows:

- Statement coverage measures whether each instruction has been executed, and branch coverage focuses on whether each branch of control structure (*e.g.*, in `if` or `switch-case` statements) has been covered, both of which are control-flow-based criteria.
- Data-flow coverage [46] enforces the coverage of each variable definition and its uses to detect data-flow anomalies.
- Model-based coverage criteria [3, 52] aim to cover more program behaviors via abstracted behavior models. Other comprehensive variants of test coverage could be referred to [2].

However, none of these criteria can be directly applied to test DNNs due to its unique architecture, as explained below.



(a) (b)

Figure 1: (a) An example of a fully connected DNN. (b) Behaviors of DNNs and relations between defined coverage criteria (the red points denote erroneous behaviors therein).

2.2 Deep Neural Network Architecture

In our paper, we regard a DL system as any software system that includes one or more DNNs.[1] Unlike traditional software, programmed with deterministic algorithms by developers, DNNs are programmed by the training data, selected features, and network structures (*e.g.*, number of layers). Specially, a DNN consists of multiple interconnected neurons organized on layers: the *input* layer, the *output* layer, and one or multiple *hidden* layers. Each *neuron* is a computing unit that computes its output by applying an *activation function* to its input. In classic DNNs, each neuron is fully-connected with all neurons on the next layer, and each edge has a *weight*, which indicates the strength of the connections between neurons. Overall, a DNN could be considered as a function that transforms a given input to the output, and this function is decided by the aggregated effects from its computation units (*i.e.*, neurons), each of which contributes to the whole computation procedure. Figure 1(a) shows an example of a three-layer DNN.

To accomplish a task (*e.g.*, prediction on the autonomous vehicles' steering angle by monitored images), DNNs are trained and programmed through a large set of labelled training data. However, similar to traditional software, DNNs may also contain defects (*e.g.*, give wrong steering angles) due to incorrect, incomplete training data, or even the wrongly stipulated run-time programming (*i.e.*, training) procedure. For example, human analyst may include erroneous and noisy data when collecting training data. In such a case, a given input data might be wrongly handled (*e.g.*, classified, predicted), causing losses and even severe tragedies, if the flawed DNNs are deployed to safety-critical systems (*e.g.*, the recent Tesla autonomous driving accident[2]). For the complex and high-dimensional real-world inputs, it is almost impossible for human to ensure all possible, even corner-case data are included. To systematically test and uncover hidden defects of DNNs, it is crucial to define a set of suitable coverage criteria for evaluating the test adequacy as well as gauging the internal covered states of DNNs to gain confidence on the testing results of DNNs.

[1] In particular, a DL system may either be entirely composed of DNNs, or have DNNs as its core with extra software encapsulation. In this paper, we mostly focus on DNNs since it is the core of a DL system, and our methods could be extended to support general DL systems. Although the training program of the current state-of-the-art DNNs are still written as traditional software, the obtained DNN from the training program is fundamentally different in how the logic is encoded.

[2] http://www.bbc.com/news/world-us-canada-43604440

3 COVERAGE CRITERIA FOR TESTING DL SYSTEMS

For traditional software testing, developers design and seek a set of representative test data from the whole large input space, hoping that the selected test data could detect the software defects under limited computational resources.[3]

Testing coverage criteria is proposed to shatter and approximate the software internal states. It partitions the input space and establishes the relation of an input subspace and an approximated software internal state. In this way, compared with the test data from a single input subspace, the same number of test data from different input sub-spaces would have a higher chance to cover more diverse software states, resulting in a higher possibility to detect more diverse software defects. Over the past decades, a set of well-designed coverage criteria [2] (*e.g.*, statement coverage, branch coverage) have demonstrated their practical value and are widely adopted in software industry to systematically guide the testing process to unveil the software defects at different levels, *e.g.*, (1) *Unit level*: testing small snippets of functions. (2) *Integration level*: testing multiple sub-modules or functions to check their interactions. (3) *System level*: testing the software system as a whole.

The current state-of-the-practice DNN testing, however, is still at its early stage and mainly relies on the prediction accuracy (similar to black-box system level testing that only observes inputs and its corresponding outputs), lacking systematic testing coverage criteria for defect detection. Furthermore, traditional software and DNNs have obvious differences, so existing coverage criteria for traditional software could not be directly applied to DNNs.

In this section, we design a set of DNN testing coverage criteria from multiple levels, aiming to gauge the testing adequacy of DNNs and facilitate the detection of those erroneous behaviors from multiple portrayals. To be useful towards industry level applications, we believe that the test criteria should be simple, scalable as well general enough to be applied to a large range of DNNs without confining on specific DNN structure or activation functions. Conceptually, similar to traditional software, the behaviors of DNNs can be divided into two categories, *i.e.*, major function behaviors and corner-case behaviors, both of which may contain erroneous behaviors (see Figure 1(b) and our evaluation results in Section 4). We have taken these factors into consideration during the design of coverage criteria.

Let $N = \{n_1, n_2, \ldots\}$ be a set of neurons of a DNN. Let $T = \{\mathbf{x}_1, \mathbf{x}_2, \ldots\}$ be a set of test inputs. We use $\phi(\mathbf{x}, n)$ to denote a function that returns the output of a neuron $n \in N$ under a given test input $\mathbf{x} \in T$.[4] Let the DNN have l layers and L_i denote the set of neurons on the i-th layer ($1 \leq i \leq l$).

3.1 Neuron-Level Coverage Criteria

At the neuron-level, we use the output values of neuron n determined from the training to characterize its behaviors. Since the internal logic of a DNN is mostly programmed by training data, intuitively, the functionality (*i.e.*, neuron output) for each neuron of a DNN should follow some statistical distribution that is largely determined

[3] The input space of a software could be so large that it is often impossible to enumerate and test all the possibilities given limited computation resource.

[4] This paper focuses on feedforward neural networks. For recurrent neural networks (RNNs), we can unroll a certain depth of layers of an RNN and adapt $\phi(\mathbf{x}, n)$ by setting $\mathbf{x}$ to be an input sequence.

by the training data. The output distribution of a neuron obtained from training data analysis would allow to approximately characterize the major function regions whose output values are often triggered by input data with a similar statistical distribution to the training data, and the corner cases whose output values rarely occur. However, for a practical-sized DNN, obtaining an accurate output distribution for each neuron would be computationally intensive. With the similar spirit while being scalable, we leverage the neuron output value boundaries obtained from training data to approximate the major function region and corner-case region.

Specially, for a neuron n, let high_n and low_n be its upper and lower boundary output values, respectively, on the value range of its activation function, where high_n and low_n are derived from the training dataset analysis. We refer to $[\text{low}_n, \text{high}_n]$ as the major function region of a neuron n.

Definition 3.1. For a test input $\mathbf{x} \in T$, we say that a DNN is located in its **major function region** given $\mathbf{x}$ iff $\forall n \in N : \phi(\mathbf{x}, n) \in [\text{low}_n, \text{high}_n]$.

To exhaustively cover the major function regions, we partition $[\text{low}_n, \text{high}_n]$ into k sections, and require each of them to be covered by the test inputs. We name this coverage as k-multisection neuron coverage.

(i) k-multisection Neuron Coverage. Given a neuron n, the k-multisection neuron coverage measures how thoroughly the given set of test inputs T covers the range $[\text{low}_n, \text{high}_n]$. To quantify this, we divide the range $[\text{low}_n, \text{high}_n]$ into k equal sections (*i.e.*, k-multisections), for $k > 0$. We write S_i^n to denote the set of values in the i-th section for $1 \leq i \leq k$.

If $\phi(\mathbf{x}, n) \in S_i^n$, we say the i-th section is covered by the test input $\mathbf{x}$. Therefore, for a given set of test inputs T and the neuron n, its k-multisection neuron coverage is defined as the ratio of the number of sections covered by T and the total number of sections, *i.e.*, k in our definition. We define the k-multisection coverage of a neuron n as:

$$\frac{|\{S_i^n \mid \exists \mathbf{x} \in T : \ \phi(\mathbf{x}, n) \in S_i^n\}|}{k}.$$

We further define the k-multisection neuron coverage of a DNN as:

$$\text{KMNCov}(T, k) = \frac{\sum_{n \in N} |\{S_i^n \mid \exists \mathbf{x} \in T : \ \phi(\mathbf{x}, n) \in S_i^n\}|}{k \times |N|}.$$

However, for a neuron n, there are also cases where $\phi(\mathbf{x}, n)$ may locate out of $[\text{low}_n, \text{high}_n]$, *i.e.*, $\phi(\mathbf{x}, n) \in (-\infty, \text{low}_n)$ or $\phi(\mathbf{x}, n) \in (\text{high}_n, +\infty)$. We refer to $(-\infty, \text{low}_n) \cup (\text{high}_n, +\infty)$ as the corner-case region of a neuron n.

Definition 3.2. For a test input $\mathbf{x} \in T$, we say that a DNN is located in its **corner-case region** given $\mathbf{x}$ iff $\exists n \in N : \phi(\mathbf{x}, n) \in (-\infty, \text{low}_n) \cup (\text{high}_n, +\infty)$.

Note that the profiled outputs of a neuron obtained from the training data would not locate into the corner-case region.In other words, if test inputs follow a similar statistical distribution with the training data, a neuron output would rarely locate in corner-case region as well. Nevertheless, it does not mean that testing the corner cases of a neuron is not important because defects of DNNs could also locate in the corner-case regions as demonstrated in Section 4.3.

To cover these corner-case regions of DNNs, we define two coverage criteria, *i.e.*, neuron boundary coverage and strong neuron activation coverage. Given a test input $\mathbf{x}$, if $\phi(\mathbf{x}, n)$ belongs to $(-\infty, \text{low}_n)$ or $(\text{high}_n, +\infty)$, we say the corresponding corner-case region is covered. To quantify this, we first define the number of covered corner-case regions as follows:

$$\text{UpperCornerNeuron} = \{n \in N \mid \exists \mathbf{x} \in T : \ \phi(\mathbf{x}, n) \in (\text{high}_n, +\infty)\};$$
$$\text{LowerCornerNeuron} = \{n \in N \mid \exists \mathbf{x} \in T : \ \phi(\mathbf{x}, n) \in (-\infty, \text{low}_n)\}.$$

(ii) Neuron Boundary Coverage. Neuron boundary coverage measures how many corner-case regions (*w.r.t.* both of the upper boundary and the lower boundary values) have been covered by the given test input set T. It is defined as the ratio of the number of covered corner cases and the total number of corner cases ($2 \times |N|$):

$$\text{NBCov}(T) = \frac{|\text{UpperCornerNeuron}| + |\text{LowerCornerNeuron}|}{2 \times |N|}.$$

Some recent research on DNNs interpretability empirically shows that the hyperactive neurons might potentially deliver useful learning patterns within DNNs [30, 61]. Based on this intuition, the proposed coverage criteria in the rest of this section focus more on the hyperactive neuron cases (*e.g.*, top-k neuron coverage in the next subsection). Similar to neuron boundary coverage, we further define strong neuron activation coverage to measure the coverage status of upper-corner cases.

(iii) Strong Neuron Activation Coverage. Strong neuron activation coverage measures how many corner cases (*w.r.t.* the upper boundary value high_n) have been covered by the given test inputs T. It is defined as the ratio of the number of covered upper-corner cases and the total number of corner cases ($|N|$):

$$\text{SNACov}(T) = \frac{|\text{UpperCornerNeuron}|}{|N|}.$$

3.2 Layer-Level Coverage Criteria

At layer-level, we use the top hyperactive neurons and their combinations (or the sequences) to characterize the behaviors of a DNN.

For a given test input $\mathbf{x}$ and neurons n_1 and n_2 on the same layer, we say n_1 is more active than n_2 given $\mathbf{x}$ if $\phi(\mathbf{x}, n_1) > \phi(\mathbf{x}, n_2)$. For the i-th layer, we use $\text{top}_k(\mathbf{x}, i)$ to denote the neurons that have the largest k outputs on that layer given $\mathbf{x}$. For example, in Figure 1(a), assume $\phi(\mathbf{x}, n_1)$ and $\phi(\mathbf{x}, n_3)$ are larger than $\phi(\mathbf{x}, n_2)$, the top-2 neurons on layer 1 are n_1 and n_3 (depicted in green).

(i) Top-k Neuron Coverage. The top-k neuron coverage measures how many neurons have once been the most active k neurons on each layer. It is defined as the ratio of the total number of top-k neurons on each layer and the total number of neurons in a DNN:

$$\text{TKNCov}(T, k) = \frac{|\bigcup_{\mathbf{x} \in T}(\bigcup_{1 \leq i \leq l} \text{top}_k(\mathbf{x}, i))|}{|N|}.$$

The neurons from the same layer of a DNN often play similar roles and the top active neurons from different layers are important indicators to characterize the major functionality of a DNN. Intuitively, to more thoroughly test a DNN, a test dataset should uncover more top active neurons.

(ii) Top-k Neuron Patterns. Given a test input $\mathbf{x}$, the sequence of the top-k neurons on each layer also forms a pattern. For example, in Figure 1(a), assume the neurons in green are the top-2 neurons on each

layer, the pattern can be represented as $(\{n_1, n_3\}, \{n_5, n_6\}, \{n_8, n_9\})$. More formally, a pattern is an element of $2^{L_1} \times 2^{L_2} \times \cdots \times 2^{L_l}$, where 2^{L_i} is the set of subsets of the neurons on i-th layer, for $1 \leq i \leq l$. Given the test input set T, the number of top-k neuron patterns for T is defined as:

$$\text{TKNPat}(T, k) = |\{(\text{top}_k(\mathbf{x}, 1), \ldots, \text{top}_k(\mathbf{x}, l)) \mid \mathbf{x} \in T\}|.$$

Intuitively, the top-k neuron patterns denote different kinds of activated scenarios from the top hyperactive neurons of each layer.

4 EXPERIMENTS

We implement *DeepGauge* on Keras 2.1.3 [10] with TensorFlow 1.5.0 backend [1], and apply the proposed testing criteria to DNNs for evaluation in this section.

4.1 Evaluation Subjects

Datasets and DNN Models. We select two popular publicly-available datasets, *i.e.*, MNIST [32] and ImageNet [42] (see Table 1) for evaluation. MNIST is for handwritten digits recognition, containing 70,000 input data in total, of which 60,000 are training data and 10,000 are test data. On MNIST, we use three pre-trained LeNet family models (LeNet-1, LeNet-4, and LeNet-5) [32] for analysis.

To further demonstrate the usefulness of our criteria towards larger scale real-world DL systems, we also select ImageNet, a large set of general image dataset (*i.e.*, ILSVRC-2012 [42]) for classification containing more than 1.4 million training data and 50,000 test data from 1,000 categories. The DNNs we used for ImageNet are pre-trained VGG-19 [44] and ResNet-50 [24] models, both of which are relatively large in size and obtain competitive records in the ILSVRC competition [42], containing more than 16,000 and 94,000 neurons, and 25 and 176 layers, respectively. As a DNN testing criterion towards future industry level application, we believe the scalability up-to ImageNet-like or even larger data size and model size is almost indispensable.

Adversarial Test Input Generation. Besides using original test data accompanied in the corresponding dataset for coverage evaluation, we further explore four state-of-the-art adversarial test input generation techniques (*i.e.*, FGSM [20], BIM [31], JSMA [37], and CW [8]) for comparative study. Each of the adversarial techniques generates tests to detect DNN's potential defects through the minor perturbations on a given input, described as follows:

- FGSM crafts adversarial examples using loss function $J(\Theta, \mathbf{x}, y)$ with respect to the input feature vector, where Θ denotes the model parameters, $\mathbf{x}$ is the input, and y is the output label of $\mathbf{x}$, the adversarial example is generated as: $\mathbf{x}^* = \mathbf{x} + \epsilon \, \text{sign}(\nabla_{\mathbf{x}} J(\Theta, \mathbf{x}, y))$.
- BIM applies adversarial noise η many times iteratively with a small parameter ϵ, rather than one η with one ϵ at a time, which gives a recursive formula: $\mathbf{x}_0^* = \mathbf{x}$ and $\mathbf{x}_i^* = \text{clip}_{\mathbf{x}, \epsilon}(\mathbf{x}_{i-1}^* + \epsilon \, \text{sign}(\nabla_{\mathbf{x}_{i-1}^*} J(\Theta, \mathbf{x}_{i-1}^*, y)))$, where $\text{clip}_{\mathbf{x}, \epsilon}(\cdot)$ denotes a clipping of the values of the adversarial sample such that they are within an ϵ-neighborhood of the original input $\mathbf{x}$.
- JSMA is proposed for targeted misclassification. For an input $\mathbf{x}$ and a neural network F, the output of class j is denoted as $F_j(\mathbf{x})$. To achieve a target misclassification class t, $F_t(\mathbf{x})$ is increased while the probabilities $F_j(\mathbf{x})$ of all other classes $j \neq t$ decrease, until $t = \arg\max_j F_j(\mathbf{x})$.
- Carlini/Wagner (CW): Carlini and Wagner recently proposed new optimization-based attack technique which is arguably the most effective in terms of the adversarial success rates achieved with minimal perturbation [8]. In principle, the CW attack is to approximate the solution to the following optimization problem:

$$\arg\min_{\mathbf{x}^*} \quad \lambda L(\mathbf{x}, \mathbf{x}^*) - J(\Theta, \mathbf{x}^*, y),$$

  where L is a loss function to measure the distance between the prediction and the ground truth, and the constant λ is to balance the two loss contributions. In this paper, we adopt the CW_∞, where each pixel is allowed to be changed by up to a limit.[5]

Figure 2 shows examples of the generated tests of the four adversarial techniques on the sampled data from MNIST test set. In this example, we could see that compared with FGSM and BIM, JSMA and CW perturb fewer pixels on the sampled test input. Furthermore, given the same input data but different DNNs, the same technique would often generate different adversarial test results. For example, given the input image 7, JSMA generates different results on DNNs (*i.e.*, LeNet-1, LeNet-4 and LeNet-5). In other words, the studied adversarial techniques are often DNN dependent.

4.2 Evaluation Setup

MNIST. On MNIST dataset, each image is single-channel of size $28 \times 28 \times 1$. Before the evaluation starts, we first obtain the DNN's neuron output statistical information through runtime profiling each of the studied DNNs (*i.e.*, LeNet-1, LeNet-4, and LeNet-5) using the 60,000 training data. When testing evaluation starts, for each DNN under analysis, we run the 10,000 test data on the model to obtain the corresponding coverage. For each studied DNN, we further generate another four sets of adversarial test data which can explore defects of DL,[6] through FGSM [20], BIM [31], JSMA [37], and CW [8]. We show that *DeepGauge* is general and easy to be tested on the state-of-the-art adversarial test generation techniques.

After generating the four adversarial datasets, we aggregate each of them with the original MNIST test dataset (with a total size 20,000 for each), which enables us to perform the comparative study on how the adversarial test data enhances the defect detection ability from our coverage criteria measurement. Since the studied adversarial test generation techniques are model dependent, the adversarial datasets generated by the same adversarial techniques are actually different for each model, though the same number (*i.e.*, five) datasets are used to evaluate on each model. For each adversarial technique, we actually use it to generate three adversarial datasets, one for each of LeNet-1, LeNet-4, and LeNet-5, respectively.

The detailed parameter configurations for each criterion are shown in Table 2, where u and l are the output upper bound (maximal value) and lower bound (minimal value) obtained for each neuron during profiling, respectively; σ is the standard deviation of the outputs of a neuron during profiling. Although the definition of neuron boundary coverage and strong neuron activation coverage are based on u and l alone (*i.e.*, neuron output UpperBound (UB) and LowerBound (LB)), it would also be interesting to see what results could be obtained if we further tighten corner-case regions (*i.e.*, increase upper bound

[5] The paper [8] suggests that L_∞ could be successful to change the classification of an image to a desired label by the lowest bit of each pixel.

[6] Each generated adversarial dataset is of the same size as the original test set.

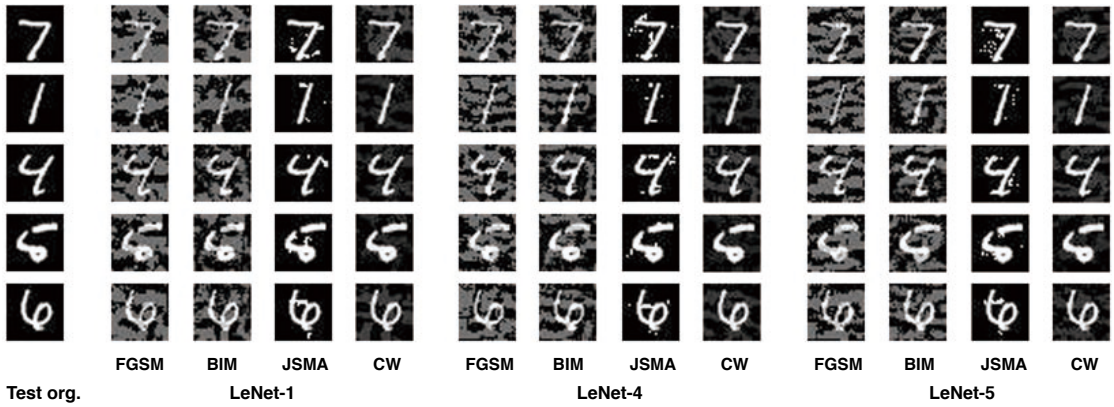


Figure 2: Examples of original sampled test data from MNIST in comparison to the ones generated by each adversarial technique on the corresponding studied DNN models.

Table 1: Breakdowns of datasets and DNN models.

DataSet	Dataset Description	DNN Model	#Neuron	#Layer	Test Data Source for Eval.
MNIST	Digit recog.	LeNet-1	52	7	Test org.& FGSM/BIM/JSMA/CW
		LeNet-4	148	8	
		LeNet-5	268	9	
ImageNet	General image with 1000-classes	VGG-19	16,168	25	Test org.& FGSM/BIM/CW
		ResNet-50	94,059	176	

Table 2: The parameter configurations for coverage evaluation.

DL Coverage Criteria	Parameter Configuration		
k-multisection Neuron Cov. (KMNC)	k=1,000	k=10,000	N.A.
Neuron Boundary Cov. (NBC)	LB=l	LB=$l - 0.5 * \sigma$	LB=$l - \sigma$
	UB=u	UB=$u + 0.5 * \sigma$	UB=$u + \sigma$
Strong Neuron Activation Cov. (SNAC)	UB=u	UB=$u + 0.5 * \sigma$	UB=$u + \sigma$
Top-k Neuron Cov. (TKNC)	k=1	k=2	k=3
Top-k Neuron Patterns (TKNP)	k=1	k=2	k=3

and decrease lower bound). Therefore, besides setting UB and LB to u and l as defined in Section 3.1, we also evaluate another two configurations by increasing UB (resp. decreasing LB) by $0.5 * \sigma$ and σ for neuron boundary coverage and strong neuron activation coverage (see Table 2). In total, we have 3 (models)×5 (datasets)×14 (criterion settings) = 210 evaluation configurations for MNIST.

ImageNet. ImageNet (ILSVRC-2012) [42] is more challenging for evaluation due to its large data size (more than 1.4 million training data) as well as large image size ($224 \times 224 \times 3$) for processing. Moreover, the DNNs that achieve high accuracy are often complex. Compared with LeNet family models, the studied VGG-19 and ResNet-50 are much more complex in terms of both neurons and layers. Due to the computational complexity of adversarial test generation on ImageNet for analysis, we randomly sample images from each of its labeled categories in the original ImageNet test dataset, with a total number of 5,000 images as the test data for our evaluation. We also try to use FGSM, BIM, JSMA, and CW to generate adversarial tests for each of the studied DNNs. However, we are unable to set up JSMA to run successfully on either of the two DNNs.[7] Overall, we have a total of 2 (models)×4 (datasets)×14 (criterion settings) = 112 experimental configurations for ImageNet.

To support such large scale evaluation, we run the experiments on a computer cluster. Each cluster node runs a GNU/Linux system with Linux kernel 3.10.0 on a 18-core 2.3GHz Xeon 64-bit CPU with 196 GB of RAM and also an NVIDIA Tesla M40 GPU with 24G. The whole experiments still take months to complete especially for the adversarial test data generation process.

[7] This could be potentially caused by the massive data size and the complexity of VGG-19 and ResNet-50. The similar issue on JSMA was also reported in a previous work [57].

4.3 Experimental Results

In our experiments, we have seen useful testing feedbacks from multiple perspectives with each testing criterion, showing some unique portrayal of the runtime behavior of DNNs. We first describe some obtained results and then summarize our findings.[8]

4.3.1 MNIST and ImageNet. **MNIST.** As shown in Table 3, the coverage of different criteria obtained by the adversarial techniques generally increase compared with the original MNIST test dataset. For instance, as for LeNet-4, the JSMA increases the coverage of the original tests from 39.7% to 52.3% by 31.7% in 10,000-multisection neuron coverage, from 9.1% to 16.2% by 78% in neuron boundary coverage, from 13.5% to 27.7% by 105% in strong neuron activation coverage, from 62.2 to 66.2 by 6.6% in top-1 neuron coverage, and from 787 to 1,395 by 77.3% in top-1 neuron patterns.

The increase of coverage infers that the adversarial test data overall explore new DNNs' internal states, some of which are not covered by the original tests. As such adversarial test data reveal defects of studied LeNet models, it indicates that generating tests towards improving the coverage of the proposed criteria might potentially trigger more states of a DNN, incurring higher chances of defect detection, which is consistent with the usage of test coverage in traditional software testing.

ImageNet. The testing coverage (Table 4) on the ImageNet shares some similarity with MNIST data while showing some differences. VGG-19 and ResNet-50 models are much larger in size and complexity, potentially causing the obtained coverage lower than that

[8] Due to the page limit, we put more detailed experimental result discussion, as well as data plot on the paper's accompanying website https://deepgauge.github.io/.

Table 3: Coverage results of *DeepGauge* on MNIST, LetNet family models, and generated test data by adversarial techniques.

Testing Criteria	DNN	Eval. Config.	Test org.	Test org. + FGSM	Test org. +BIM	Test org. +JSMA	Test org. +CW
KMNC (%)	LN-1	k=1,000	64.5	74.8	68.1	77.7	72.8
		k=10,000	37.3	48.6	46.6	51.5	49.5
	LN-4	k=1,000	70.4	75.5	73.6	77.7	74.5
		k=10,000	39.7	49.7	50.2	52.3	50.1
	LN-5	k=1,000	68.5	72.0	71.5	73.8	71.2
		k=10,000	37.2	46.0	47.6	48.8	46.8
NBC (%)	LN-1	LB=l,UB=u	43.3	47.1	49.0	46.2	44.2
		l-0.5*σ,u+0.5*σ	17.3	21.2	21.2	18.3	17.3
		l-σ,u+σ	8.7	8.7	8.7	9.6	8.7
	LN-4	LB=l,UB=u	9.1	12.2	13.9	16.2	10.5
		l-0.5*σ,u+0.5*σ	0.7	1.0	1.0	2.7	1.0
		l-σ,u+σ	0.3	0.3	0.3	1.0	0.3
	LN-5	LB=l,UB=u	8.6	10.5	11.6	13.4	9.1
		l-0.5*σ,u+0.5*σ	1.7	2.1	1.9	3.0	2.0
		l-σ,u+σ	1.1	1.3	1.3	1.7	1.1
SNAC (%)	LN-1	UB=u	38.5	42.3	46.2	42.3	40.4
		UB=u+0.5*σ	23.1	25.0	28.9	25.0	23.1
		UB=u+σ	17.3	17.3	17.3	19.2	17.3
	LN-4	UB=u	13.5	16.2	18.9	27.7	13.5
		UB=u+0.5*σ	1.4	1.4	1.4	5.4	1.4
		UB=u+σ	0.7	0.7	0.7	2.0	0.7
	LN-5	UB=u	14.9	16.4	20.2	23.5	14.9
		UB=u+0.5*σ	3.4	4.1	3.7	6.0	3.4
		UB=u+σ	2.2	2.6	2.6	3.4	2.2
TKNC (%)	LN-1	k=1	61.5	61.5	61.5	63.5	61.5
		k=2	86.5	88.5	88.5	88.5	86.5
		k=3	92.3	92.3	92.3	92.3	92.3
	LN-4	k=1	62.2	63.5	64.9	66.2	64.9
		k=2	79.1	79.7	80.4	81.8	80.4
		k=3	85.1	87.2	87.8	86.5	86.5
	LN-5	k=1	49.3	53.7	53.7	51.9	52.2
		k=2	63.8	66.8	67.5	66.0	66.0
		k=3	72.4	73.9	74.6	74.3	74.6
TKNP	LN-1	k=1	76	77	77	100	86
		k=2	915	1,271	1,185	1,325	1,270
		k=3	3,716	6,069	5,708	6,597	5,823
	LN-4	k=1	787	1,210	1,140	1,395	1,389
		k=2	6,190	11,185	11,268	12,140	11,742
		k=3	9,301	18,515	18,491	18,356	18,194
	LN-5	k=1	1,136	2,141	1,775	2,031	2,011
		k=2	6,947	13,987	12,614	12,797	12,456
		k=3	9,684	19,361	19,215	19,201	19,157

of LeNet in many cases. Consider the 10,000-multisection neuron coverage, FGSM achieves 48.6% on LeNet-1, but only 18.8% on VGG-19. At first glance, it is tempting to draw the conclusion that a DNN with higher complexity is more difficult to be covered by tests. Our results show that this might not be generally applicable. For example, the original tests achieves 22.8% KMNC ($k = 10,000$) on ResNet-50, but only obtains 13.5% on VGG-19, although ResNet-50 is more complex in terms of the number of neurons and layers. Compared with MNIST, the generated adversarial tests on ImageNet incur even higher increase on the neuron boundary coverage (NBC) and strong neuron activation coverage (SNAC) (Table 4). For example, on ResNet50 under LB=l and UB=u configuration, BIM increases these two criteria by 280% (from 4.1% to 11.5%) and 279% (from 4.7% to 13.1%), respectively.

The top-1 neuron coverage obtained by both MNIST and ImageNet (see Table 3 (TKNC) and Table 4 (TKNC)) show that many neurons of a DNN have been triggered into a top-k (*i.e.*, 1, 2, and 3 in our evaluated cases) hyperactive states. For example, on VGG-19, the sampled original tests of ImageNet achieve 58.8% top-1 neuron coverage, and 81.6% top-3 neuron coverage. Although the top-k coverage improvement is not that obvious compared with other criteria, the adversarial data still trigger more neurons as top-k activated neurons in many cases, which detects the hidden defects.

Table 4: Coverage results of *DeepGauge* on ImageNet, VGG-19 and ResNet-50, and generated tests by adversarial techniques.

Testing Criteria	DNN	Eval. Config.	Test org.	Test org. + FGSM	Test org. +BIM	Test org. +CW
KMNC (%)	VGG-19	k=1,000	32.2	36.9	38.0	35.7
		k=10,000	13.5	18.8	19.1	18.5
	ResNet-50	k=1,000	43.0	47.5	47.8	47.4
		k=10,000	22.8	29.3	29.6	29.4
NBC (%)	VGG-19	LB=l,UB=u	2.8	8.7	7.4	2.9
		l-0.5*σ, u+0.5*σ	1.5	4.0	3.4	1.5
		l-σ, u+σ	1.1	3.2	2.5	1.1
	ResNet-50	LB=l,UB=u	4.1	6.9	11.5	4.7
		l-0.5*σ, u+0.5*σ	1.5	2.1	6.0	1.7
		l-σ, u+σ	0.9	1.2	3.9	0.9
SNAC (%)	VGG-19	UB=u	4.6	10.5	9.8	4.7
		UB=u+0.5*σ	3.0	8.0	6.8	3.1
		UB=u+σ	2.1	6.3	5.1	2.2
	ResNet-50	UB=u	4.7	7.0	13.1	5.4
		UB=u+0.5*σ	2.1	2.8	8.3	2.4
		UB=u+σ	1.3	1.8	6.1	1.4
TKNC (%)	VGG-19	k=1	58.8	61.5	68.1	68.7
		k=2	74.3	76.2	80.8	81.2
		k=3	81.6	82.9	85.9	85.9
	ResNet-50	k=1	26.8	30.3	29.1	29.5
		k=2	36.0	38.6	38.3	38.3
		k=3	42.3	44.7	44.3	44.3
TKNP	VGG-19	k=1	4,999	8,265	9,989	9,816
		k=2	4,999	9,581	9,998	9,816
		k=3	4,999	9,921	9,998	9,816
	ResNet-50	k=1	4,999	9,998	9,998	9,948
		k=2	4,999	9,998	9,998	9,948
		k=3	4,999	9,998	9,998	9,948

For different test input datasets, it is often the case that only a fixed subset of neurons of each layer would function as top hyperactivated neurons. This would be a hint that the top hyperactivated neurons of each layer might be able to describe the high-level major function skeleton of a neuron network. In comparison with the top-k neuron patterns (see Table 3 (TKNP) and Table 4 (TKNP)), albeit most of the top hyperactive neurons are relatively stable for each layer, their combination still captures the structural difference of input data.[9] These two layer-level criteria altogether provide us with the information on which neurons matter the most within each layer; and the top-k neuron patterns would mostly be able to differentiate the input data when k is properly selected given a target DNN. The findings indicate that generating tests to cover more top-k neuron patterns would have a higher chance to find defects of a DNN.

4.3.2 Findings and Remarks. The overall experimental results demonstrate the usefulness of our proposed testing criteria for DNNs and are also helpful to explain the difference of the state-of-the-practice adversarial techniques form multiple perspectives:

- The original test data of MNIST and ImageNet cover both the DNNs major function region (see KMNC) as well as corner-case region (see NBC and SNAC) in Tables 3 and 4. This also happens to the generated adversarial datasets, showing that a defect of DNN can occur either in a major function region or a corner-case region, both of which should be extensively tested.
- The test data generated by 4 studied adversarial techniques (combined with original test data) generally boost the coverage measured by our criteria. Since an adversarial test data could potentially reveal defects of a DL system, it means that boosting the

[9] Our in-depth investigation on the generated top-k neuron patterns for ImageNet show that the relatively large pattern coverage improvement (even for the top-1 case) is relevant to the large # of neurons and layers in VGG-19 and ResNet-50.

coverage of our testing criteria could to some extent enhance the fault detection ability, which is consistent with the practical purpose of testing criteria widely adopted in traditional software testing. It also shows that our test criteria metrics could capture the DNNs' internal behavioral difference of benign and adversarial test data. We note that increasing the test coverage does not necessarily imply that new defects could be detected in traditional software testing. The same conclusion applies to our coverage criteria for DNNs as well, though better-defined coverage criteria would be much more pronounced in finding defects.

- Test data (including the generated test data by adversarial) evaluated on both MNIST and ImageNet mostly obtain a higher k-multisection neuron coverage than the neuron boundary coverage and strong neuron activation coverage, revealing that the test data cover more of the major function region than the corner-case region of a DNN. The design of future DL testing techniques should also take into account the covering of the corner-case region.
- For most of the evaluated configurations, we find that a higher strong neuron activation coverage is more achieved than its corresponding neuron boundary coverage. This might be caused by the unique characteristics of those activation functions in our studied DNNs, which makes the lower region (small value) more difficult to be covered than the upper (large value) region of the statistical profiling distribution.[10] This observation is consistent with the models we studied, as LeNet family, VGG-19, and ResNet-50 all use `ReLU` as activation functions, which could make the lower regions of a neuron much smaller than the upper regions.[11]

Remark 1. In general, for neuron boundary coverage and strong neuron activation coverage, the higher (resp. lower) the neuron's upper (resp. lower) bound, the less increment on coverage we observe; for top-k neuron coverage, the larger the value of k, the less increment on coverage; for top-k neuron patterns, the larger the value of k, the more increment on patterns.

Remark 2. For neuron boundary coverage and strong neuron activation coverage, the 4 adversarial techniques have sufficient diversity on the performance, which is similar to traditional test cases which aim to cover different potential defects. Specifically, we observe that the adversarial tests generated by CW are harder to be distinguished by the test coverage since the CW perturbation concentrates more on the objects with smaller magnitude. In many cases, adversarial tests generated by CW contain minimal perturbation, which may trigger less internal behavior changes of DNNs.

4.4 Comparison with DeepXplore's Neuron Coverage (DNC)

Pei *et al*. [38] propose a kind of neuron activation coverage as the measurement for testing data diversity of a DNN and argue that the higher the activation coverage, the more states of a DNN could be explored, with a higher chance for defects detection. A key parameter of DNC is a user-specified threshold, and if an output of a neuron is larger than the threshold, the neuron is counted as covered. To demonstrate the difference between our set of criteria and DNC, we set up the DNC evaluation with the same dataset, model, as well as adversarial data generation settings as described in Section 4.1. For the threshold parameter, we first set thresholds to be 0 and 0.75, as used in [38]; to make an even more comprehensive comparison, we also use two other settings (*i.e*., 0.2 and 0.5). Figures 3 and 4 show that the results of DNC obtained on the original test dataset and the dataset generated by adversarial techniques for MNIST and ImageNet are almost the same for all experimental settings, indicating that DNC is unable to differentiate the original test data from adversarially generated ones, which trigger the correct and incorrect behaviors of a DNN, respectively. This means that DNC could hardly capture the difference between original test data and corresponding test data generated by adversarial techniques. However, to detect the defects of DNNs in a more fine-grained level, it is necessary that the coverage criteria capture such minor differences, where the defects (adversarially triggered states) also lie in.

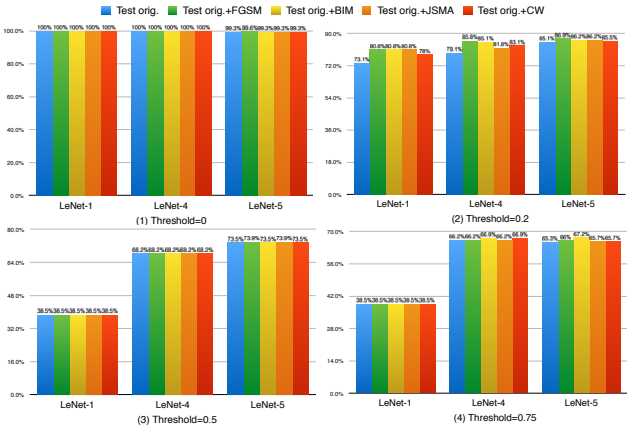


Figure 3: The DeepXplore neuron coverage results on MNIST dataset under different threshold configurations.

Our further in-depth investigation on DNC reveals that, this coverage criterion imposes several limitations: (1) DNC uses the same threshold as the activation evaluation for all the neurons. However, we find that the output statistical distribution of different neurons are quite diverse. Given a test suite for analysis, the outputs of some neurons may exhibit quite a small variance with a large mean value, while others might have a large variance with a low mean value. Therefore, using the same threshold for all neurons without considering the disparity in neuron's functional distributions would greatly diminish the accuracy. For example, given a neuron with very small mean and standard deviation, even a slightly larger user-specified threshold would generally determine that this neuron cannot be covered. (2) DNC normalizes the dynamic range of neuron outputs according to max and min output of neurons on the corresponding layer for each input image under analysis. This raises an issue that the same normalized activation value (*e.g*., 0.3) means differently

[10] LeNet-1 is the only exceptional case, and this might be caused by the over-simplicity of its network (see Table 1).

[11] In particular, `ReLU` function propagates the positive output of a neuron to the next layer while blocking the negative output by setting it to zero, which stops influencing its following layers.

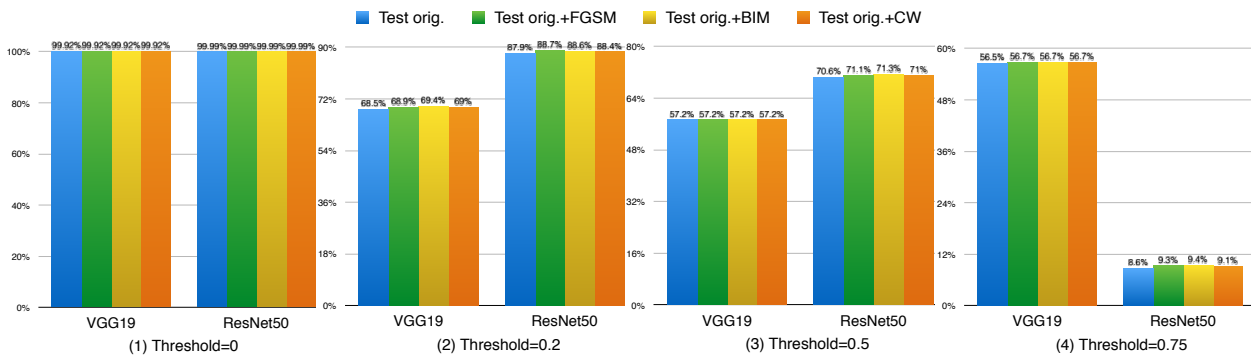


Figure 4: The DeepXplore neuron activation coverage results on ImageNet dataset for different threshold settings.

for different input data, as the max and min output of each layer might change for each input, which renders the notion of "larger than a given threshold" inconsistent among different inputs.

Doing so will also eliminate the relativity in activation magnitude among different inputs, which is a very important property to support the findings of neuron activation coverage. We instead specify the upper and lower bounds for each neuron obtained from the analysis on the training dataset, as opposed to each individual input. In other words, our method relies on the statistics of the training set which is used to determine the main functionality of the DNN system.

4.5 Threats to Validity and Discussion

The selection of evaluation subjects (*i.e.*, dataset and DNN models) could be a threat to validity. We try to counter this by using the commonly-studied MNIST dataset and the practical large-scale dataset ImageNet; for each studied dataset, we use the well-known pre-trained models of different sizes and complexity ranging from 52 neurons up to more than 90,000 neurons. Even though, some of our results might not generalize to other datasets and DNN models. Another threat could be caused by the configurable hyper-parameters in the coverage criteria definition. As a countermeasure while considering the limited computational resources, we evaluate each criterion with different settings, and analyze the influence of the parameters on criteria accuracy. Even though, it might still not cover the best parameter use-cases. For example, our evaluation studied $k = 1,000$ and $k = 10,000$ for k-multisection neuron coverage. We leave the optimized hyper-parameter selection in our future work. Further threat could be caused by the quality of training data used for distribution (*i.e.*, the interval range) analysis of neuron output. In this paper, we consider publicly available well-pretrained DNN models accompanied by training data with good quality.

For adversarial test generation, we select four popular state-of-the-practice techniques to simulate defects from different sources and granularity. We either follow the authors' suggested settings or use their default settings. Moreover, to make comprehensive comparisons with DeepXplore's neuron coverage (DNC), we evaluate DNC with multiple threshold settings.

5 RELATED WORK

In this section, we attempt to review the most relevant work in three aspects: testing, verification, and security of DL systems.

5.1 Testing of DL Systems

Traditional practices in measuring machine learning systems mainly rely on probing their accuracy on test inputs which are randomly drawn from manually labeled datasets and *ad hoc* simulations [54]. However, such black-box testing methodology may not be able to find various kinds of corner-case behaviors that may induce unexpected errors [19]. Wicker *et al.* [53] recently proposed a Scale Invariant Feature Transform feature guided black-box testing and showed its competitiveness with CW and JSMA along this direction.

Pei *et al.* [38] proposed a white-box differential testing algorithm for systematically finding inputs that can trigger inconsistencies between multiple DNNs. They introduced neuron coverage for measuring how much of the internal logic of a DNN has been tested.However, it still exhibits several caveats as discussed in Section 4.4. DeepTest [49] investigates a basic set of image transformations (*e.g.*, scaling, shearing, and rotation) from OpenCV and shows that they are useful to detect defects in DNN-driven autonomous cars. Along this direction, DeepRoad [59] uses input image scene transformation and shows its potentiality with two scenes (*i.e.*, snowy and rainy) for autonomous driving testing. The scene transformation is obtained through training a generative adversarial network (GAN) with a pair of collected training data that cover the statistical features of the two target scenes.

Compared with traditional software, the dimension and potential testing space of a DNN is often quite large. DeepCT [35] adapts the concept of combinatorial testing, and proposes a set of coverage based on the neuron input interaction for each layer of DNNs, to guide test generation towards achieving reasonable defect detection ability with a relatively small number of tests. Inspired by the MC/DC test criteria in traditional software [29], Sun *et al.* [47] proposed a set of four adapted MC/DC test criteria for DNNs, and show that generating tests guided by the proposed criteria on small scale neural networks (consisting of `Dense` layers with no more than 5 hidden layers and 400 neurons) exhibits higher defect detection ability than random testing. However, whether MC/DC criteria scale to real-world-sized DL systems with multiple types of layers

still needs further investigation. Instead of observing the runtime internal behaviors of DNNs, DeepMutation [34] proposes to mutate DNNs (*i.e.*, injecting faults either from the source level or model level) to evaluate the test data quality, which could potentially be useful for test data prioritization in respect of robustness on a given DNN.

Our work of proposing multi-granularity testing coverage for DL systems is mostly orthogonal to the existing work. Compared with the extensive study on traditional software testing, testing DL is still at an early stage. Most existing work on DL testing lacks some suitable criteria to understand and guide the test generation process. Since test generation guided by coverage criteria (*e.g.*, statement coverage, branch coverage) towards the exploration of diverse software states for defect detection has become the *de facto* standard in traditional software testing [5, 16, 17, 33], the study to design suitable testing criteria for DL is desperately demanding. This paper makes an early attempt towards this direction by proposing a set of testing criteria. Our criteria not only can differentiate state-of-the-art adversarial test generation techniques, but also potentially be useful for the measurement of test suite diversity by analyzing the DNNs' internal states from multiple portrayals. We believe that our proposed criteria set up an important cornerstone and bring a new opportunity to design more effective automated testing techniques guided by testing criteria for DL systems.

5.2 Verification of DL Systems

Formal methods can provide formal guarantees about safety and robustness of verified DL systems [22, 28, 39, 40, 50, 53]. The main concern of formal methods are their scalability for real-world-sized (*e.g.*, $100,000$ neurons or even more) DL systems.

The early work in [40] provided an abstraction-refinement approach to checking safety properties of multi-layer perceptrons. Their approach has been applied to verify a network with only 6 neurons. DLV [53] can verify local robustness of DL systems w.r.t. a set of user specified manipulations. Reluplex [28] is a sound and complete SMT-based approach to verifying safety and robustness of DL systems with `ReLU` activation functions. The networks verified by Reluplex in [28] have 8 layers and 300 `ReLU` nodes. DeepSafe [22] uses Reluplex as its verification engine and has the same scalability problem as Reluplex. AI^2 [50] is a sound analyzer based on abstract interpretation that can reason about safety and robustness of DL systems. It trades precision for scalability and scales better than Reluplex. The precision of AI^2 depends on abstract domains used in the verification, and it might fail to prove a property when it actually holds. VERIVIS [39] can verify safety properties of DL systems when attackers are constrained to modify the inputs only through given transformation functions. However, real-world transformations can be much more complex than the transformation functions considered in the paper.

5.3 Attacks and Defenses of DL Systems

A plethora of research has shown that deep learning systems can be fooled by applying carefully crafted adversarial perturbation added to the original input [6–9, 20, 48, 55, 60], many of which are based on gradient or optimization techniques. However, it still lacks extensive study on how these adversarial techniques differentiate in terms of DNNs' internal states. In this study, we make an early attempt towards such a direction based on our proposed criteria.

With the rapid development of adversarial attack techniques, extensive studies have been performed to circumvent adversarial attacks. Galloway *et al*. [18] recently observe that low-precision DNNs exhibit improved robustness against some adversarial attacks. This is primarily due to the stochastic quantization in neural network weights. Ensemble adversarial training [51], GAN based approaches [43, 45], random resizing and random padding [56], game theory [13], and differentiable certificate [41] methods are all investigated to defend against adversarial examples. By applying image transformations, such as total variance minimization and image quilting, very effective defenses can be achieved when the network is trained on the aforementioned transformed images [23]. For more extensive discussion on current state-of-the-art defense techniques, we refer readers to [4].

Our proposed testing criteria enable the quantitative measurement of different adversarial attack techniques from the software engineering perspective. This could be potentially helpful for understanding and interpreting DNNs' behaviors, based on which more effective DNN defense technique could be designed. In future work, it would be also interesting to examine how to integrate the proposed testing criteria into the DL development life cycle towards building high quality DL systems.

6 CONCLUSION AND FUTURE WORK

The wide adoption of DL systems, especially in many safety-critical areas, has posed a severe threat to its quality and generalization property. To effectively measure the testing adequacy and lay down the foundation to design effective DL testing techniques, we propose a set of testing criteria for DNNs. Our experiments on two well-known datasets, five DNNs with diverse complexity, and four state-of-the-art adversarial testing techniques show that the tests generated by the adversarial techniques incur obvious increases of the coverage in terms of the metrics defined in the paper. This demonstrates that *DeepGauge* could be a useful indicator for evaluating testing adequacy of DNNs.

To the best of our knowledge, our work is among the early studies to propose testing criteria for DL systems. We expect that the proposed testing criteria could be particularly amenable to DL testing in the wild. In the next step, we will continue to explore alternative testing criteria for DNNs, such as the combination of both hyperactive and hypoactive neurons. We also plan to study the proposed testing criteria guided automated test generation techniques for DNNs. We hope that our study not only provides an avenue to illuminate the nature and mechanism of DNNs, but also lays the foundation towards understanding and building generic and robust DL systems.

ACKNOWLEDGMENTS

This work was partially supported by National Key R&D Program of China 2017YFC1201200 and 2017YFC0907500, Fundamental Research Funds for Central Universities of China AUGA5710000816, JSPS KAKENHI Grant 18H04097. We gratefully acknowledge the support of NVIDIA AI Tech Center (NVAITC) to our research. We also appreciate the anonymous reviewers for their insightful and constructive comments.

REFERENCES

[1] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 265–283.

[2] Paul Ammann and Jeff Offutt. 2008. *Introduction to Software Testing* (1 ed.). Cambridge University Press, New York, NY, USA.

[3] Cyrille Artho, Quentin Gros, Guillaume Rousset, Kazuaki Banzai, Lei Ma, Takashi Kitamura, Masami Hagiya, Yoshinori Tanabe, and Mitsuharu Yamamoto. 2017. Model-Based API Testing of Apache ZooKeeper. In *2017 IEEE Int. Conf. on Software Testing, Verification and Validation (ICST 2017)*. Tokyo, Japan, 288–298.

[4] Anish Athalye, Nicholas Carlini, and David Wagner. 2018. Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*. https://arxiv.org/abs/1802.00420

[5] Benoit Baudry and Martin Monperrus. 2015. The Multiple Facets of Software Diversity: Recent Developments in Year 2000 and Beyond. *ACM Computing Surveys (CSUR)* 48, 1 (2015), 16.

[6] Wieland Brendel, Jonas Rauber, and Matthias Bethge. 2018. Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models. In *ICLR*.

[7] Nicholas Carlini and David Wagner. 2017. Adversarial Examples are not Easily Detected: Bypassing Ten Detection Methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. ACM, 3–14.

[8] Nicholas Carlini and David Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. In *IEEE Symposium on Security and Privacy*. 39–57.

[9] Sen Chen, Minhui Xue, Lingling Fan, Shuang Hao, Lihua Xu, Haojin Zhu, and Bo Li. 2018. Automated Poisoning Attacks and Defenses in Malware Detection Systems: An Adversarial Machine Learning Approach. *Computers & Security* 73 (2018), 326–344.

[10] François Chollet et al. 2015. Keras. https://github.com/fchollet/keras.

[11] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. 2012. Multi-column Deep Neural Networks for Image Classification. In *CVPR*. 3642–3649.

[12] Dan Ciresan, Alessandro Giusti, Luca M Gambardella, and Jürgen Schmidhuber. 2012. Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images. In *NIPS*. 2843–2851.

[13] Guneet S. Dhillon, Kamyar Azizzadenesheli, Jeremy D. Bernstein, Jean Kossaifi, Aran Khanna, Zachary C. Lipton, and Animashree Anandkumar. 2018. Stochastic Activation Pruning for Robust Adversarial Defense. In *ICLR*.

[14] Arilo C. Dias Neto, Rajesh Subramanyan, Marlon Vieira, and Guilherme H. Travassos. 2007. A Survey on Model-based Testing Approaches: A Systematic Review. In *Proc. 1st ACM Int'l Workshop on Empirical Assessment of Software Engineering Languages and Technologies*. 31–36.

[15] ECSS. 2009. Space engineering - Software.

[16] Robert Feldt, Richard Torkar, Tony Gorschek, and Wasif Afzal. 2008. Searching for Cognitively Diverse Tests: Towards Universal Test Diversity Metrics. In *Proceedings of the 2008 IEEE International Conference on Software Testing Verification and Validation Workshop*. IEEE, 178–186.

[17] Gordon Fraser and Andrea Arcuri. 2013. Whole Test Suite Generation. *IEEE Trans. Softw. Eng.* 39, 2 (Feb. 2013), 276–291. https://doi.org/10.1109/TSE.2012.14

[18] Angus Galloway, Graham W. Taylor, and Medhat Moussa. 2018. Attacking Binarized Neural Networks. In *ICLR*.

[19] Ian Goodfellow and Nicolas Papernot. 2017. The Challenge of Verification and Testing of Machine Learning.

[20] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *ICLR*.

[21] Google Accident. 2016. A Google Self-driving Car Caused a Crash for the First Time. https://www.theverge.com/2016/2/29/11134344/google-self-driving-car-crash-report

[22] Divya Gopinath, Guy Katz, Corina S. Pasareanu, and Clark Barrett. 2018. DeepSafe: A Data-driven Approach for Checking Adversarial Robustness in Neural Networks. *International Symposium on Automated Technology for Verification and Analysis (ATVA)* (2018).

[23] Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens van der Maaten. 2018. Countering Adversarial Images using Input Transformations. In *ICLR*.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *CVPR*. 770–778.

[25] Warren He, Bo Li, and Dawn Song. 2018. Decision Boundary Analysis of Adversarial Examples. In *ICLR*.

[26] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. 2012. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine* 29, 6 (2012), 82–97.

[27] Yue Jia and Mark Harman. 2011. An Analysis and Survey of the Development of Mutation Testing. *IEEE Trans. Softw. Eng.* 37, 5 (Sept. 2011), 649–678.

[28] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. *International Conference on Computer Aided Verification (CAV)* (2017).

[29] Hayhurst Kelly J., Veerhusen Dan S., Chilenski John J., and Rierson Leanna K. 2001. *A Practical Tutorial on Modified Condition/Decision Coverage*. Technical Report.

[30] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, et al. 2018. Interpretability beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors (TCAV). In *International Conference on Machine Learning*. 2673–2682.

[31] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2017. Adversarial Examples in the Physical World. *ICLR* (2017).

[32] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based Learning Applied to Document Recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[33] Lei Ma, Cyrille Artho, Cheng Zhang, Hiroyuki Sato, Johannes Gmeiner, and Rudolf Ramler. 2015. GRT: Program-Analysis-Guided Random Testing (T). In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE) (ASE '15)*. IEEE Computer Society, Washington, DC, USA, 212–223. https://doi.org/10.1109/ASE.2015.49

[34] Lei Ma, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Fei Juefei-Xu, Chao Xie, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang. 2018. DeepMutation: Mutation Testing of Deep Learning Systems. *International Symposium on Software Reliability Engineering (ISSRE)* (2018).

[35] Lei Ma, Fuyuan Zhang, Minhui Xue, Bo Li, Yang Liu, Jianjun Zhao, and Yadong Wang. 2018. Combinatorial Testing for Deep Learning Systems. *arXiv preprint arXiv:1806.07723* (2018).

[36] Glenford J. Myers, Corey Sandler, and Tom Badgett. 2011. *The Art of Software Testing* (3rd ed.). Wiley Publishing.

[37] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The Limitations of Deep Learning in Adversarial Settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE, 372–387.

[38] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Deepxplore: Automated Whitebox Testing of Deep Learning Systems. In *Proceedings of the 26th Symposium on Operating Systems Principles*. 1–18.

[39] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Towards Practical Verification of Machine Learning: The Case of Computer Vision Systems. *CoRR* abs/1712.01785 (2017). arXiv:1712.01785 http://arxiv.org/abs/1712.01785

[40] Luca Pulina and Armando Tacchella. 2010. An Abstraction-Refinement Approach to Verification of Artificial Neural Networks. In *International Conference on Computer Aided Verification*. Springer, 243–257.

[41] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. 2018. Certified Defenses against Adversarial Examples. In *ICLR*.

[42] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *IJCV* 115, 3 (2015), 211–252.

[43] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. 2018. Defense-GAN: Protecting Classifiers Against Adversarial Attacks Using Generative Models. In *ICLR*.

[44] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-scale Image Recognition. *ICLR* (2015).

[45] Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. 2018. PixelDefend: Leveraging Generative Models to Understand and Defend against Adversarial Examples. In *ICLR*.

[46] Ting Su, Ke Wu, Weikai Miao, Geguang Pu, Jifeng He, Yuting Chen, and Zhendong Su. 2017. A Survey on Data-Flow Testing. *ACM Computing Surveys (CSUR)* 50, 1, Article 5 (March 2017), 35 pages.

[47] Y. Sun, X. Huang, and D. Kroening. 2018. Testing Deep Neural Networks. *ArXiv e-prints* (March 2018). arXiv:cs.LG/1803.04792

[48] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing Properties of Neural Networks. In *ICLR*.

[49] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars. In *International Conference on Software Engineering (ICSE)*. ACM.

[50] Dana Drachsler-Cohen Petar Tsankov Swarat Chaudhuri Martin Vechev Timon Gehr, Matthew Mirman. 2018. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *IEEE Symposium on Security and Privacy (SP)*.

[51] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. 2018. Ensemble Adversarial Training: Attacks and Defenses. In *ICLR*.

[52] Mark Utting and Bruno Legeard. 2007. *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[53] Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. 2018. Feature-Guided Black-Box Safety Testing of Deep Neural Networks. *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)* (2018).

[54] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. 2016. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann.

[55] Chaowei Xiao, Jun-Yan Zhu, Bo Li, Warren He, Mingyan Liu, and Dawn Song. 2018. Spatially Transformed Adversarial Examples. In *ICLR*.

[56] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. 2018. Mitigating Adversarial Effects through Randomization. In *ICLR*.

[57] Weilin Xu, David Evans, and Yanjun Qi. 2018. Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks. In *Network and Distributed System Security Symposium (NDSS)*.

[58] Fangyi Zhang, Jürgen Leitner, Michael Milford, Ben Upcroft, and Peter Corke. 2015. Towards Vision-based Deep Reinforcement Learning for Robotic Motion Control. *arXiv:1511.03791* (2015).

[59] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid. 2018. DeepRoad: GAN-based Metamorphic Autonomous Driving System Testing. *ArXiv e-prints* (Feb. 2018). arXiv:cs.SE/1802.02295

[60] Zhengli Zhao, Dheeru Dua, and Sameer Singh. 2018. Generating Natural Adversarial Examples. In *ICLR*.

[61] Bolei Zhou, Yiyou Sun, David Bau, and Antonio Torralba. 2018. Revisiting the Importance of Individual Units in CNNs via Ablation. *arXiv preprint arXiv:1806.02891* (2018).

[62] Hong Zhu, Patrick A. V. Hall, and John H. R. May. 1997. Software Unit Test Coverage and Adequacy. *ACM Computing Survey* 29, 4 (Dec. 1997), 366–427.