Revisiting Neuron Coverage for DNN Testing: A Layer-Wise and Distribution-Aware Criterion

Yuanyuan Yuan, Qi Pang, and Shuai Wang*

The Hong Kong University of Science and Technology, Hong Kong, China
{yyuanaq, qpangaa, shuaiw}@cse.ust.hk

Abstract—Various deep neural network (DNN) coverage criteria have been proposed to assess DNN test inputs and steer input mutations. The coverage is characterized by neurons having certain outputs, or the discrepancy between neuron outputs. Nevertheless, recent research indicates that neuron coverage criteria show little correlation with test suite quality.

In general, DNNs approximate distributions, by incorporating hierarchical layers, to make predictions for inputs. Thus, we champion to deduce DNN behaviors based on its approximated distributions from a layer perspective. A test suite should be assessed using its induced layer output distributions. Accordingly, to fully examine DNN behaviors, input mutation should be directed toward diversifying the approximated distributions.

This paper summarizes eight design requirements for DNN coverage criteria, taking into account distribution properties and practical concerns. We then propose a new criterion, NEURAL COVERAGE (NLC), that satisfies all design requirements. NLC treats a single DNN layer as the basic computational unit (rather than a single neuron) and captures four critical properties of neuron output distributions. Thus, NLC accurately describes how DNNs comprehend inputs via approximated distributions. We demonstrate that NLC is significantly correlated with the practical concerns. We then propose a new criterion, NEURAL COVERAGE (NLC), that satisfies all design requirements. NLC is referred to as "neural coverage" rather than a single neuron and captures the diversity of a test suite across a number of tasks (classification, generation, and data formats (image and text)). Its capacity to discover DNN prediction errors is promising. Test input mutation guided by NLC results in a greater quality and diversity of exposed erroneous behaviors.

I. INTRODUCTION

Recent work has incorporated software testing methodologies to successfully evaluate DNN [44, 45, 50]. However, fundamental understanding of a DNN's mechanism has remained largely unexplored due to its intrinsic non-linearity. As a result, DNN evaluation is highly dependent on the test suite's representativeness. Testing criteria are thus required to characterize a test suite's quality and toward which the input mutations can comprehensively explore DNN behaviors.

Similar to how coverage criteria are developed in software testing, recent research proposes several neuron coverage criteria. In principle, the current criteria are mostly designed to monitor the outputs of neurons in a DNN model. They argue that each neuron is an "individual computing unit" (as the authors put it [33]), or "individually encoding a feature" [28], comparable to a statement in traditional software. Further, despite neuron outputs are continuous, existing criteria often first convert neuron outputs into discrete states (e.g., activated or not) to reduce the modeling complexity. Thus, by analyzing the discrete states of neurons, the DNN coverage exercised by an input is measured. Recent works measure the distance between neuron output traces of a new input and historical inputs to decide the discrepancy of the new input. The derived coverage is calculated via clusters [32] or discretized buckets over some distance scores [22, 23].

However, the key assumption — each neuron is an individual computing unit — often does not hold true for modern DNNs. Given a DNN is composed of stacked and non-linear layers, the intermediate outputs (also known as latent representations) from neurons are continuous and highly entangled. Disentangling intermediate outputs remains a long-lasting challenge [26, 27, 35]. Recent research also points out that the efficacy of a test suite to uncover faults and existing neuron coverage attained by the test suite are not always positively correlated [16].

Establishing proper DNN coverage criteria requires a principled understanding about how DNN performs computations, which is fundamentally different from traditional software. A DNN, as a composition of non-linear functions (i.e., layers), extracts task-oriented features (e.g., classification vs. localization) and then makes decisions. Features, that are hierarchically propagated by layer outputs, approximate the posterior distribution of underlying explanatory factors $H$ for an input $x$, namely $P(H|x)$ [4, 5]. For example, an image classifier generates a set of posterior distributions $P(h|x)$ (where $i$ is the layer index and $h_i \in H$) with each $h_i$ depicts class-invariant properties of an image from different aspects (e.g., shape, texture) [4]. In addition, the dependability of a DNN is defined as the degree to which the distribution, approximated on training data, is accurate when applied to real test data.

This work initializes a principled view on designing DNN coverage criteria, by assessing test inputs regarding their induced layer output distributions. We first concretize our focus into eight design requirements. These requirements characterize the distribution approximated by a DNN from various aspects, and also take practical constraints (e.g., cost, hyper-parameter free) into account. Then, we propose a new coverage criterion, namely NLC (Neural Coverage), that can fully meet the design requirements. NLC is referred to as "neural coverage" rather than neuron coverage since we focus on the overall activity of groups of neurons (i.e., each layer) and consider relations of neurons (different with all existing criteria; see comparison in Sec. III-B). Our evaluation includes nine DNNs that analyze various data formats (e.g., image, text) and perform various tasks (e.g., classification, generation). We show that NLC is...
This work advocates to design DNN testing criteria via Y. We show that NLC can better assess the quality of test neural labeled as in DNN refers to 5 as a range of continuous in DNN may perspective [40]. And with the extracted features getting more abstract at deeper layers, the ending layer returns the probability that the input belongs to each label. Sequential data, such as text, is processed by recurrent models (e.g., LSTM) in a similar way.

**Generative Models.** Another popular paradigm is the deep generative model, which estimates the joint probability distribution \( P(X, Y) \) of data X and label Y. The most common models are VAE [24] and GAN [15]. Generative models are primarily adopted to synthesize realistic data. To do so, both VAE and GAN project the data distribution to a simple distribution (e.g., normal distribution). To generate realistic outputs, random inputs sampled from the pre-defined distribution are fed into deep generative models.

**Visualizing DNN Outputs.** We empirically assess and visualize the distribution of intermediate neurons’ outputs. We prepare a DNN (denoted as Net; see implementation in [1]) with two neurons \( n_1 \) and \( n_2 \) (thus the dimensions are two) in the last layer and extend Net into 1) a VAE, 2) a class-conditional VAE (C-VAE), and 3) a classifier. The three models are trained on the MNIST dataset [12] for 1) random image generation, 2) labeled image (e.g., “2”) generation, and 3) image classification. In short, these models inherit the same structure from Net yet perform different tasks by using different trailing layers.

![Fig. 1: Visualization of \( n_1 \) and \( n_2 \)’s outputs in three models extended from Net. X- and y-axis are outputs of \( n_1 \) and \( n_2 \).](https://github.com/Yuanyuan-Yuan/NeuraL-Coverage [1].)

Fig. 1 shows the outputs of \( n_1 \) and \( n_2 \) in these models. Each dot denotes one output of \( n_1 \) and \( n_2 \), corresponds to one input. Colors represent labels of the inputs (e.g., red represents “9”). In Fig. 1(a) and Fig. 1(c), neuron outputs are jointly distributed corresponding to different labels, whereas in Fig. 1(b), outputs are not split. Also, outputs gather as regions of varied density and shapes. Fig. 1 demonstrates that, models sharing the same architecture, inputs, and output range (e.g., Fig. 1(a) Fig. 1(b)), may have distinct behaviors, which can hardly be assessed using range or distance of neuron outputs by existing criteria.

This study captures DNN behaviors by examining the continuity of neuron outputs. It also considers neuron entanglement, where neurons in a layer are correlated and jointly form distributions that reflect how DNNs (of different tasks) understand inputs distinctly.

**Neural Coverage.** Distributions formed by entangled neurons are important indicators of DNN behaviors. Given that “neural” holistically, correlates to distributions of neuron outputs when understanding input features, this work proposes Neural Coverage, NLC, a new criterion that captures properties in distributions formed by neuron outputs. The term “neural,” as an adjective, better reflects that neuron outputs are processed in the continuous form and “L” also indicates each Layer is considered as the minimal computing unit.

DNNs are often explained from the distribution approximation perspective [4, 5, 40]. Following, this section provides a high-level overview of DNNs to motivate NLC.

**“Neural” and “Network”**. The term neural in DNN may be obscure. In cognitive science, neural signifies non-linear functions (e.g., Sigmoid) in DNNs that simulate how human neurons process signals [6, 37]. DNNs are powerful at analyzing high-dimensional media data due to their non-linearity [3, 4].

Neurons in a layer are therefore highly entangled, impeding to comprehend the DNN’s internal mechanics. Moreover, outputs of neurons are continuous. The term network in DNN refers to topological connections of non-linear layers. Each layer has a group of neurons cooperating to process certain input features.

The DNN hierarchy is established by gradually abstracting features from shallow to deep layers. For instance, image classifiers commonly collect shapes in an image at shallow layers and texture at deeper layers [14, 49]. Each layer (not each neuron) serves a minimal computing unit, whose output distribution encodes its focused features.

Given a collection of data X labeled as Y, most popular DNNs can be categorized as discriminative and generative models, depending on how X and Y are involved in the distribution approximation.

**Discriminative Models.** Deep discriminative models approximate the conditional probability distribution \( P(Y|X) \). A classification model requires Y as discrete labels, and a regression model (e.g., forecasting) requires Y as a range of continuous values. For instance, a convolutional neural network (CNN) can classify images by extracting features (conveyed via outputs of intermediate neurons) and making decisions in one forward propagation. A layer outputs a conditional distribution denoting the extracted input features at this layer [4, 5, 40]. And with the diversity and capacity of a test suite to reveal DNN defects. Meanwhile, by taking NLC as the fuzzing feedback, we show that more and diverse errors can be found, and the mutations guided by NLC generate more natural inputs. Our contributions are summarized as:

- This work advocates to design DNN testing criteria via a principled understanding about how DNN, with its hierarchical layers, approximate distributions. We argue that a proper DNN testing criterion should model the continuous neuron outputs and neuron entanglement, instead of analyzing each neuron individually.
- We propose eight design requirements by considering the distributions approximated by a DNN and practical implementation considerations. We accordingly design NLC, satisfying all these requirements.
- We show that NLC can better assess the quality of test suites. NLC is particularly correlated with the diversity and fault-revealing capacity of a test suite. Fuzz testing under the guidance of NLC can also detect a large number of errors, which are shown as diverse and natural-looking.

**Artifact:** We released our artifact and supplementary materials at https://github.com/Yuanyuan-Yuan/NeuraL-Coverage [1].

II. PRELIMINARY AND OVERVIEW

**Fig. 1: Visualization of \( n_1 \) and \( n_2 \)’s outputs in three models extended from Net. X- and y-axis are outputs of \( n_1 \) and \( n_2 \).**

Given a collection of data X labeled as Y, most popular DNNs can be categorized as discriminative and generative models, depending on how X and Y are involved in the distribution approximation.

**Discriminative Models.** Deep discriminative models approximate the conditional probability distribution \( P(Y|X) \). A classification model requires Y as discrete labels, and a regression model (e.g., forecasting) requires Y as a range of continuous values. For instance, a convolutional neural network (CNN) can classify images by extracting features (conveyed via outputs of intermediate neurons) and making decisions in one forward propagation. A layer outputs a conditional distribution denoting the extracted input features at this layer [4, 5, 40]. And with the diversity and capacity of a test suite to reveal DNN defects. Meanwhile, by taking NLC as the fuzzing feedback, we show that more and diverse errors can be found, and the mutations guided by NLC generate more natural inputs. Our contributions are summarized as:

- This work advocates to design DNN testing criteria via a principled understanding about how DNN, with its hierarchical layers, approximate distributions. We argue that a proper DNN testing criterion should model the continuous neuron outputs and neuron entanglement, instead of analyzing each neuron individually.
- We propose eight design requirements by considering the distributions approximated by a DNN and practical implementation considerations. We accordingly design NLC, satisfying all these requirements.
- We show that NLC can better assess the quality of test suites. NLC is particularly correlated with the diversity and fault-revealing capacity of a test suite. Fuzz testing under the guidance of NLC can also detect a large number of errors, which are shown as diverse and natural-looking.

**Artifact:** We released our artifact and supplementary materials at https://github.com/Yuanyuan-Yuan/NeuraL-Coverage [1].

II. PRELIMINARY AND OVERVIEW

DNNs are often explained from the distribution approximation perspective [4, 5, 40]. Following, this section provides a high-level overview of DNNs to motivate NLC.

**“Neural” and “Network”.** The term neural in DNN may be obscure. In cognitive science, neural signifies non-linear functions (e.g., Sigmoid) in DNNs that simulate how human neurons process signals [6, 37]. DNNs are powerful at analyzing high-dimensional media data due to their non-linearity [3, 4].

Neurons in a layer are therefore highly entangled, impeding to comprehend the DNN’s internal mechanics. Moreover, outputs of neurons are continuous. The term network in DNN refers to topological connections of non-linear layers. Each layer has a group of neurons cooperating to process certain input features. The DNN hierarchy is established by gradually abstracting features from shallow to deep layers. For instance, image classifiers commonly collect shapes in an image at shallow layers and texture at deeper layers [14, 49]. Each layer (not each neuron) serves a minimal computing unit, whose output distribution encodes its focused features.

Given a collection of data X labeled as Y, most popular DNNs can be categorized as discriminative and generative models, depending on how X and Y are involved in the distribution approximation.

**Discriminative Models.** Deep discriminative models approximate the conditional probability distribution \( P(Y|X) \). A classification model requires Y as discrete labels, and a regression model (e.g., forecasting) requires Y as a range of continuous values. For instance, a convolutional neural network (CNN) can classify images by extracting features (conveyed via outputs of intermediate neurons) and making decisions in one forward propagation. A layer outputs a conditional distribution denoting the extracted input features at this layer [4, 5, 40]. And with the diversity and capacity of a test suite to reveal DNN defects. Meanwhile, by taking NLC as the fuzzing feedback, we show that more and diverse errors can be found, and the mutations guided by NLC generate more natural inputs. Our contributions are summarized as:

- This work advocates to design DNN testing criteria via a principled understanding about how DNN, with its hierarchical layers, approximate distributions. We argue that a proper DNN testing criterion should model the continuous neuron outputs and neuron entanglement, instead of analyzing each neuron individually.
- We propose eight design requirements by considering the distributions approximated by a DNN and practical implementation considerations. We accordingly design NLC, satisfying all these requirements.
- We show that NLC can better assess the quality of test suites. NLC is particularly correlated with the diversity and fault-revealing capacity of a test suite. Fuzz testing under the guidance of NLC can also detect a large number of errors, which are shown as diverse and natural-looking.

**Artifact:** We released our artifact and supplementary materials at https://github.com/Yuanyuan-Yuan/NeuraL-Coverage [1].
III. MOTIVATION AND RELATED WORKS

This section, by 1) comparing traditional software and DNN, 2) demonstrating limitations of prior criteria, and 3) incorporating practical considerations, proposes **eight requirements** for DNN coverage criteria that motivates NLC.

A. DNN Testing and Software Testing

**Discrete vs. Continuous States.** Software coverage is often measured by counting statements or paths executed by an input. Software testing aims to maximize code coverage, as high coverage suggests comprehensive testing. Inspired by software testing, previous DNN testing often converts a neuron’s continuous outputs [28, 33], or distance between neuron output traces [22, 23], into discrete states. The coverage is denoted as the activated states of a test input. As shown in Fig. 1, discrete states are unexpressive for describing neuron behavior. Instead, we advocate that a DNN coverage criterion should **precisely describe the continuous output of a neuron.**

**Statement vs. Correlation.** It is inaccurate to compare neurons to software statements, because each neuron is *not* an independent computing unit. In general, an “unactivated” neuron (defined by [33]) can still affect the DNN activities and final predictions. In contrast, software statements may not be covered by an input. Given that said, Fig. 1 shows that neuron outputs are often correlated (e.g., increasing simultaneously). Neuron correlation is, to some extent, analogous to program dependency, e.g., statements within an if block execute together. Rather than studying individual neurons, it is preferable (and more efficient) to measure DNN coverage by **characterizing correlations of neurons.**

**Execution vs. Estimation.** Software executes its input on an execution path. DNNs, as described in Sec. II, use a series of layers to approximate certain distributions for an input. Different DNN inputs produce different distributions, as seen in Fig. 1. Hence, we advocate to define the coverage criterion over approximated distributions. While it’s difficult to precisely describe the approximated distribution [34, 36, 48], certain informative properties of a distribution can be captured. Also, since coverage criteria are used to assess test suite quality and guide input mutations, knowing the exact distribution may not be required (see our empirical results in Sec. VI). In brief, we advocate that the DNN coverage criterion should **analyze how outputs of neurons in a layer are distributed.**

B. Existing Neuron Coverage and Their Limitations

We illustrate existing DNN coverage criteria in Fig. 2 using two neurons to ease reading. In general, prior coverage criteria leverages discretization to decide if a state is “covered.”

**Major Behavior of One Neuron.** Neuron Coverage (NC) [33], as the first criterion, argues that the major behavior of a neuron is mostly reflected when its outputs are above zero. As in Fig. 2(a), NC deems a neuron as activated if its output is above threshold $T$ ($T$ is zero or a small value above). Following, three criteria were proposed [28], namely K-Multisection Neuron Coverage (KMNC), Neuron Boundary Coverage (NBC) and Strong Neuron Activation Coverage (SNAC). We display KMNC, NBC and SNAC in Fig. 2(b), Fig. 2(c) and Fig. 2(d).

All three criteria first run training data to scope the output range of each neuron. Then, KMNC splits each range into $K$ segments and regards coverage as #segments covered by neuron outputs. NBC denotes coverage as #neuron outputs lying outside the range, while SNAC counts coverage as #neurons whose outputs are larger than the upper bound of its normal value range. Their intuition is that the major behaviors of neurons are denoted as outputs that lie in the range decided by training data.

**Limitations:** Considering the blue region in Fig. 1(c), it is easy to see that when feeding inputs of label “1”, NC over those two neurons will saturate. This indicates that criteria, which simply discretize neuron outputs, cannot properly describe the diversity of test suites (e.g., inputs of different labels). Moreover, since these criteria do not consider neuron entanglement, they should not change if the orange distribution of label “7” in Fig. 1(c) is rotated for 90 degrees. That is, these criteria might hardly reflect distinctly changed DNN behaviors.

Following 1, we argue that a neuron’s major behavior can hardly be defined using coarse-grained thresholds. Instead, given that the high-density regions formed by neuron outputs in a layer (e.g., the “dots” on Fig. 2’s background) are distinct on shapes, we deem the major behavior as the shapes which reflect the majority of neuron activates. Corner-case behaviors are therefore regarded as neuron outputs lying in low-density regions. In sum, we argue that **DNN coverage should consider density of neuron output distributions.** Measuring density considers both major and corner-case behaviors of DNNs, which is desirable in DNN testing.

**Discretizing Outputs of Multiple Neurons.** Two layer-wise criteria, Top-K Neuron Coverage (TKNC) and Top-K Neuron Patterns (TKNP), were also proposed in [28]. Both criteria denote a neuron as activated if its outputs are ranked at top-$K$ among neurons in a layer. TKNC counts #neurons once been activated, whereas TKNP records #patterns formed by activated neurons. As shown in Fig. 2(e) where there are two neurons, a neuron is activated if its output is greater than another one (i.e., $K = 1$ and $n_1 > n_2$). Sign-Sign coverage (SS) [43] captures the sign change of layer-wise adjacent neurons. A neuron is activated if 1) its outputs, and outputs from its adjacent neuron,
have different signs for any pair of input $\langle p_1, p_2 \rangle$ (i.e., $n_1^l$ and $n_2^{l+1}$ in Fig. 2(f) and simultaneously, 2) outputs from all other neurons in the same layer have an identical sign for $\langle p_1, p_2 \rangle$ (i.e., $n_1^l$ and $n_2^{l+1}$ in Fig. 2(f)).

**Limitations:** Despite considering all neurons in a layer, TKNC and TKNP still treat one neuron as a minimal computing unit and are incapable of analyzing the output distribution. They are easily saturated w.r.t. inputs of one or a few classes (see Fig. 1 and Fig. 2(e)), failing to assessing the diversity of a test suite. SS is specifically designed for ReLU-DNNs (i.e., DNNs with ReLU as the non-linear function). However, since most of the neuron outputs are nearly zero-centered due to input and layer normalization in modern DNNs [21, 42], SS can be less useful for many of them. Also, while SS considers the relation of adjacent layers, since each neuron is still a separate unit in SS, the relation of layers is modeled at the neuron-level.

As noted in Sec. II, the DNN hierarchy is formed by input feature re-use (e.g., processing shape at shallow layers and then texture at deep layers [14, 49]). The hierarchy is generally encoded into layer outputs. NLC, by capturing the distribution of layer outputs, is more desirable compared to SS.

**Discretizing Distances over Traces of Neurons.** Cluster-based Coverage (CC) assesses distances between traces of neuron outputs [32]. CC counts #clusters, where two clusters are formed if, the Euclidean distance of outputs from all neurons within a layer, is larger than a threshold. Given that more clusters are formed if neuron outputs get spread, CC can be regarded as counting #regions covered by several neuron outputs; as illustrated Fig. 2(g).

Surprise adequacy (SA) metrics, namely, Likelihood SA (LSA), Distance-ratio SA (DSA) and Mahalanobis Distance SA (MDSA), assess the distance between the neuron output trace of an input and neuron output traces of all training data to decide how “surprise” the new input is [22, 23]. LSA and DSA require iterating over all traces of neuron outputs collected from training data. MDSA reduces the computation cost by caching class-conditional mean and covariance matrices estimated on training data. Surprise Coverage (SC) derived via SAs discretizes the range of SA values into buckets, and coverage is denoted as the ratio of covered buckets.

Neuron Path Coverage (NPC) [46] is based on explainable AI techniques [2]; a path is constituted by critical neurons (identified via DNN gradients) from different layers. Similar to SCs, the coverage is calculated via discretized buckets of distance values over different paths. Holistically, since more spreading neuron outputs induce higher coverage values for SCs and NPC, they can also be expressed using Fig. 2(g).

**Limitations:** CC is more desirable than prior criteria given it is more comprehensive to overlay the output distribution. Nevertheless, CC could not capture correlations among neurons. SCs/NPC have similar limitations. In addition, unlike existing gray-box coverage criteria, NPC requires white-box access (e.g., DNN weights, back-propagation gradients) to the tested DNN, which limits its application scope.

### C. Practical Considerations

**Incorporating Training Data.** The training of a DNN involves a “test-and-fix” procedure: the DNN is tuned to reduce its errors and fit better on training data. This way, training data generally reflects how DNN works inside. We argue that **DNN coverage should support incorporating prior-knowledge extracted from training data.** Existing criteria use training data to scope neuron output ranges [28]. Nevertheless, for some criteria, e.g., TKNC/TKNP; their hyper-parameter $K$ depends on model structures instead of training data. NLC can use training data to estimate distributions of neuron outputs before being used in online testing, thereby satisfying ⑦.

**Matrix-Form Computation.** Coverage criteria are often used in costly DNN testing. According to the released code of existing criteria, we notice that they are mostly written in a loop-fashion which is slow. We find that testings guided by these criteria spend most time (over 80%) on coverage calculation. We re-implemented these criteria in a matrix-fashion, i.e., the calculation is performed as a sequence of matrix operations. Given that modern DL frameworks (e.g., PyTorch) can optimize matrix operations and also use hardware optimizations [10, 29, 38], the calculation is accelerated over 100 times; see our codebase at [1]. Therefore, we advocate that **a coverage criterion should support matrix-form computation to be optimizable by modern DL frameworks and hardware.**

**Incremental Update.** DNN coverage is also typically used to guide test input mutations. Since inputs are gradually fed, we point out that **a coverage should feature efficient incremental update.** Ideally, when processing a new input, the cost for updating coverage should be a small constant. Since CC iterates over existing clusters and #clusters increases when more data are produced, it fails to fulfill ⑦. TKNP suffers from similar drawbacks. SS requires extensively comparing each neuron output with others and is thus inefficient for incremental update. Similarly, for each new input, LSC and DSC iterate over all neuron output traces of training data, resulting in a computing cost of the (#training data) × (#neurons) magnitude. Compared with other criteria (except for CC, TKNP, SS) which have a fixed small cost with a magnitude of (#neurons) for each incremental update, LSC and DSC fail to satisfy ⑦.

**User-Friendliness.** Most existing criteria rely on hyper-parameters. When using existing criteria, we find that coverage results are usually highly sensitive to hyper-parameters (see Sec. VI-A1), whereas existing works hardly provide guidance to decide hyper-parameters under different scenarios. Viewing that deciding hyper-parameters likely requires expertise (in both software testing and deep learning), we advocate that **a DNN coverage is desirable to be hyper-parameter free.**

### D. Comparison with Previous Criteria

Table I benchmarks existing DNN coverage criteria (and NLC) w.r.t. the eight requirements summarized in this section. Note that TKNC/TKNP consider the relative relation of neuron outputs and SS considers the sign relation between two neurons. They are deemed as partially satisfying ⑦. Though CC/SCs/NPC consider a trace of neurons, how one neuron
TABLE I: Benchmarking coverage criteria for DNN testing. ✓, ○, ✗ denote support, partially support, and not support. ■ denotes gray-box (only needs neuron outputs) and □ denotes white-box that requires DNN weights and gradients.

|   | I   | II  | III-A | IV  | V   | VI  | VII | VIII |
|---|-----|-----|-------|-----|-----|-----|-----|-------|
| NC [33] | ✗   | ✗   | ✓     | ✓   | ✗   | ✗   | ✓   | ✓     |
| KMNC [36] | ✗   | ✗   | ✓     | ✓   | ✗   | ✗   | ✓   | ✓     |
| NBC [28] | ✗   | ✗   | ✓     | ✓   | ✗   | ✗   | ✓   | ✓     |
| SNAC [26] | ✗   | ✗   | ✓     | ✓   | ✗   | ✗   | ✓   | ✓     |
| TKNC [28] | ✓   | ✓   | ✓     | ✓   | ✓   | ✓   | ✓   | ✓     |
| TKNP [28] | ✓   | ✓   | ✓     | ✓   | ✓   | ✓   | ✓   | ✓     |
| SS [43] | ✗   | ✗   | ✓     | ✓   | ✗   | ✗   | ✓   | ✓     |
| CC [32] | ✗   | ✗   | ✓     | ✓   | ✗   | ✗   | ✓   | ✓     |
| LSC [22] | ✗   | ✗   | ✓     | ✓   | ✗   | ✗   | ✓   | ✓     |
| DSC [23] | ✗   | ✗   | ✓     | ✓   | ✗   | ✗   | ✓   | ✓     |
| MDSC [23] | ✗   | ✗   | ✓     | ✓   | ✗   | ✗   | ✓   | ✓     |
| NPC [46] | ✓   | ✓   | ✓     | ✓   | ✓   | ✓   | ✓   | ✓     |
| NLC | ✓   | ✓   | ✓     | ✓   | ✓   | ✓   | ✓   | ✓     |

respects to others can hardly be captured. They thus fail to satisfy 2. Given that the spreading of neuron outputs is captured, they partially fulfill 3. Also, since neuron outputs get concentrated, the neuron output range overlays the high-density regions. Similarly, the cluster/merged paths in CC/NPC and likelihood in LSC reflect high and low density regions respectively. We thus regard KMNC, NBC, SNAC, CC, and LSC as partially satisfying 3. In contrast, NLC satisfies all eight requirements, as explained in Sec. IV.

IV. DESIGN OF NEURAL COVERAGE

As noted in Sec. III-A, it’s generally difficult to precisely and efficiently describe distributions approximated by DNNs [34, 36, 48]. Instead, as depicted in Fig. 3, NLC captures four key properties of distributions: divergence, correlation, shape, and density, which corresponds to the first four criteria in Table I. Sec. VI presents empirical results to show that these properties are sufficient to form a criterion that outperforms all existing works. We now elaborate on each property.

Measuring Activation in Continuous Space. Neuron outputs are continuous floating-point numbers. Instead of discretizing continuous neuron outputs, or distances between neuron output traces, NLC directly measures neuron outputs in the continuous space. We start with the case of one neuron.

We randomly select one neuron from a pre-trained ResNet50 and plot its output distribution in Fig. 4(a) using all images in its training dataset. The x-axis denotes the output value and the y-axis denotes the ratio of involved inputs. Consistent with Fig. 1, output values converge to a certain point (zero in this case). NLC directly measures how divergent (i.e., how active) the neuron output is in its continuous form by calculating the following variance:

$$\sigma_n^2 = E[(o - E[o])(o - E[o])],$$  

where $o$ is the output of a neuron $n$. Overall, variance is widely-adopted for characterizing the divergence of a collection of samples. Eq. 1 increases when output values spread along the x-axis. In contrast, if most outputs are zero, the variance will decrease, as in Fig. 4(b). Variance serves as an important descriptor of neuron output distributions. A test suite is regarded as comprehensive if it results in a high variance. Similarly, input mutations can be launched towards the direction that increases variance. Compared with previous criteria, we require no discretization, thus taking all distinct neuron outputs into account (in response to 1). In addition, Eq. 1 does not ship with hyper-parameters (respond to 3). It’s worth noting that variance is non-monotonic. That is, its value may decrease given new inputs. To alleviate this problem, we only update it when its value increases (i.e., $\Delta \sigma_n^2 = \max(0, \Delta \sigma_n^2)$). Generally, a stable variance, which stays constantly or tends to decrease, indicates that the behaviors are (nearly) fully explored.

Characterizing Neuron Correlation. As noted in Sec. II, neurons in a layer are entangled and jointly process input features. As empirically assessed in Fig. 1, neurons manifest distinct correlations under different tasks and for inputs of different classes. For example, in Fig. 1(c), neuron $n_1$ is positively (e.g., class “1”) or negatively (e.g., class “9”) correlated with neuron $n_2$.

Fig. 5 presents three different correlations of two neurons. Given that there are more than one neuron, the DNN coverage should simultaneously consider the divergence of each neuron and correlations among neurons. Eq. 1 uses variance to describe divergence of a neuron’s output, i.e., $\sigma_n^2$ and $\sigma_2^2$. Similarly, we use the covariance, a common metric for joint variability of two variables, to characterize the correlation between two neurons. Thus, the correlation between the two neurons is formulated as

$$\varsigma_{n_1,n_2} = E[(n_1 - E[n_1])(n_2 - E[n_2])] = \varsigma_{n_2,n_1}.$$  

$\varsigma_{n_1,n_2} = 0$ implies that $n_1$ and $n_2$ are not correlated, whereas $\varsigma_{n_1,n_2} > 0$ denotes a positive correlation between $n_1$ and $n_2$, i.e.,
and \( n_1 \) will increase with \( n_2 \), and vice versa. A higher \( \varphi_{n_1,n_2} \) denotes a stronger correlation. The same holds when \( \varphi_{n_1,n_2} < 0 \). Taking Eq. 1 and Eq. 2 into account, we simultaneously represent the divergence of each neuron and correlations of neurons in a unified form as

\[
\Sigma = \begin{bmatrix} \sigma^2_{n_1} & \varphi_{n_1,n_2} \\ \varphi_{n_2,n_1} & \sigma^2_{n_2} \end{bmatrix},
\]

where \( \Sigma \) is the covariance matrix of \( n_1 \) and \( n_2 \). As we show in Fig. 5, the value of \( \varphi_{n_1,n_2} \) also represents the divergence along the “correlation direction” of two neurons. Thus, we use

\[
\frac{1}{m \times m} \left\| \Sigma \right\|_1 = \frac{1}{m \times m} \sum_{j=1}^{m} \left( \sum_{i=1}^{m} |\varphi_{n_i,n_j}| \right),
\]

where \( m \) is the number of neurons and \( \varphi_{n_i,n_j} = \sigma^2_{n_i} \) to represent the DNN coverage. The size of \( \Sigma \) is \( m \times m \) and \( \frac{1}{m \times m} \) works as the normalization term to eliminate the effect of \( m \) neurons to the coverage values. In the case where more than two neurons are correlated, e.g., \( n_i, n_j \) and \( n_k \), \( \varphi \) will characterize the correlation via \( \varphi_{n_i,n_j} \) and \( \varphi_{n_j,n_k} \). Hence, Eq. 4 facilitates addressing requirements \( \odot \) and \( \odot \). Note that \( \Sigma \) can be initialized using the correlation term \( \varphi_{n_i,n_j} \) decided by training data. Also, \( \Sigma \) can be further refined in a class conditionally by extending the size to \( c \times m \times m \) where \( c \) is the number of classes. It will be calculated by first being indexed using the class labels. In both two cases, NLC fulfills the requirement of \( \odot \). Further, given a DNN which contains multiple layers, the coverage of DNN is computed as

\[
\text{NLC} = \frac{1}{I} \sum_{i=m \times m}^{1} \left\| \Sigma \right\|_1,
\]

where \( I \) is the layer index. A higher NLC indicates a more diverse test suite. Test input mutations, accordingly, are expected to accumulate towards directions that maximize NLC.

### Capturing Distribution Shape

Unlike previous works where each neuron is considered separately, we take neurons from the same layer as a group. As in Fig. 1, the output distribution of neurons in the same layer is mostly distinct under different tasks. Moreover, for the separated regions (in different colors) under certain tasks (e.g., classification), they may also have different shapes characterizing various DNN behaviors. Hence, we capture a major behavior by describing the shape of a distribution. We now explain why \( \Sigma \) in Eq. 3 captures a distribution’s shape.

Fig. 6: Transforming white noise to a desired distribution.

To ease understanding, we first show how to transform a white noise (whose covariance matrix is degenerated into an identity matrix \( I \)) to the distribution formed by outputs from a group of neurons. Fig. 6(a) shows a white noise \( \mathcal{N} \) and Fig. 6(c) presents the output distribution of two neurons \( x \) and \( y \). The transformation is thus decomposed into two elementary operators — scaling and rotating. The scaling operator spreads two neuron outputs and rotating introduces correlations between \( x \) and \( y \). Rotating and scaling are defined by rotation matrix and scaling matrix as

\[
R_{2 \times 2} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix},
S_{2 \times 2} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix},
\]

respectively. Thus, the output distribution can be represented as \( (RS)\mathcal{N} \). Following Eq. 1 and Eq. 2, we have

\[
\Sigma = \mathbb{E}[(RS\mathcal{N} - \mathbb{E}[RS\mathcal{N}])(RS\mathcal{N} - \mathbb{E}[RS\mathcal{N}])^\top] = (RS)\mathbb{E}[(\mathcal{N} - \mathbb{E}[\mathcal{N}])(\mathcal{N} - \mathbb{E}[\mathcal{N}])^\top](RS)^\top = (RS)(I(RS)^\top)(RS)^\top,
\]

which indicates that covariance matrix \( \Sigma \) in Eq. 3 encodes transformations from a white noise to a desired distribution. Therefore, we demonstrate that \( \Sigma \) captures the shape (i.e., how it is distributed) of the output distribution formed by outputs of neurons in a layer, satisfying \( \odot \).

### Responding to Density Change

Neuron outputs generally concentrate to certain points to form clusters, and the density varies in the output space. We first elaborate on what may introduce a local density change. Recall that we deem corner-case behaviors as neuron outputs lying at the low-density regions (e.g., boundary of one cluster), which are likely to trigger DNN defects. Given that these outliers typically have much lower local-density, their existence will lead to density change. Consider Fig. 7(a) where the DNN outputs in red are deemed as outliers. Nevertheless, since they are still in the output ranges formed by all green dots, SNAC and NBC will not treat them as corner-case behaviors. In fact, the new red outputs will not induce coverage increase for all previous criteria except CC and SCs, because CC and SCs capture how neuron outputs spread in the whole space. However, they require properly selected distance thresholds, which are hyper-parameters configured by users.

Also, for certain cases where neuron outputs are separated (e.g., the classifier in Fig. 1(c)), the appearance of a new major behavior (e.g., images get classified as a new class) also leads to density change. As in Fig. 7(b), the new cluster of red dots introduces a new major behavior, whereas NC possibly has no response since their values are lower than the threshold.

We show that NLC responds to density change. For Fig. 7(a), the \( \left\| \Sigma \right\|_1 \) of existing green dots is 13.2. It will be updated as 13.4 given the new red dots, which indicates that NLC responds. In contrast, since the new yellow dots stay close...
to the majority of existing outputs, NLC has no response. It’s worth mentioning that, despite the density changes only slightly, NLC has no resolution loss given its continuous form. Meanwhile, for Fig. 7(b), the $|\Sigma|_i$ will change from 13.2 to 14.6 when the cluster formed by red dots appears. We thus state that our criterion can respond to density change which is possibly introduced by corner-case behaviors and new major behaviors. This way, NLC satisfies $\mathcal{C}_3$.

Clarification on Covariance. Both MDSC and NLC rely on covariance. We, however, clarify that the two “covariances” imply distinct concepts. NLC characterizes how fresh inputs change the neuron correlation or introduce new distributions, using an actively-updated covariance. NLC incrementally updates covariance given new inputs and rollbacks the value if the new input is not desired. In contrast, the covariance in MDSC is a pre-computed constant that represents the distribution of historical neuron outputs, based on which the Mahalanobis distance can be computed from.

Implementation Considerations. As discussed above, it is easy to see that NLC supports matrix-form computing and has no hyper-parameter. Thus, NLC satisfies $\mathcal{C}_4$ and $\mathcal{C}_5$. We now clarify that NLC supports incremental computing (requirement of $\mathcal{C}_6$). We also show that NLC supports batch computation, which is also vital as DNNs generally take a batch of inputs and the batch size varies from 1 to hundreds. Let existing data be $B$, given a batch of new data $B'$, the covariance matrix $\Sigma$ (Eq. 4) of each layer can be calculated as

$$\Sigma = \left( B \times B' \times (\mu_B - \mu_B)(\mu_B - \mu_B)^T \left( |B| + |B'| \right)^{-1} \right) \times \left( \frac{1}{|B|} \sum_{B} + \frac{1}{|B'|} \times \Sigma_B \right) \right)^{-1},$$

where $\mu$ is the mean of a collection of data. Clearly, Eq. 8 does not iterate over existing data and the computing cost is in a magnitude of (#neurons); NLC thus satisfies $\mathcal{C}_7$.

V. IMPLEMENTATION AND CONFIGURATIONS

NLC is implemented using PyTorch (ver. 1.9.0) with roughly 2,300 LOC. All experiments are launched on one Intel Xeon CPU E5-2683 with 256GB RAM and one Nvidia GeForce RTX 2080 GPU. Below we briefly introduce settings of prior criteria compared with NLC. Their implementations are released in [1].

Hyper-parameters. We denote hyper-parameters of KMNC, TKNC, and TKNP as $K$ in Sec. VI. For SCs, we report the #covered buckets of discretized SA values. The bucket size and the thresholds in other criteria are denoted as $T$.

SS and NPC. SS is specifically designed for ReLU-DNN, whereas modern DNNs have various non-linear functions which are not supported by SS. Also, SS does not support matrix-form and incremental computing. Our preliminary study shows that it is computational infeasible due to the large number of neurons in modern DNN. NPC requires white-box DNN, which is not always practical in testing scenarios. In addition, its building block, the layer-wise propagation [2], is only applicable to discriminative models. We thus omit to compare with them.

VI. EVALUATION

A. Assessing Test Suite Quality

We first compare NLC with existing criteria on its effectiveness to assess the 1) diversity and 2) fault-revealing capability of test suites.

Settings & DNNs & Datasets. We evaluate NLC on both discriminative and generative models. The involved models and datasets are listed in Table II and Table III. For discriminative models, we choose VGG16, ResNet50, and MobileNetV2, which represent standard large-scale DNNs with sequential and non-sequential topological structures and popular models on mobile platforms. The three models are trained on two datasets of different scales, namely, CIFAR10 [25] and ImageNet [11].

Models trained on CIFAR10 have over 92% test accuracy and those trained on ImageNet are provided by PyTorch. CIFAR10 is composed of 32×32 images labeled as ten classes. ImageNet contains images of 1,000 classes and the average image size is 469×387. We also evaluate NLC on the text model, i.e., a LSTM [19] trained on the IMDB dataset [30] for sentiment analysis [13]. Given that each layer in the LSTM is unrolled to process sequential inputs, the unrolled layers are considered separately for computing coverage [44]. For generative models, we choose the SOTA model BigGAN. The BigGAN is also trained on CIFAR10 and ImageNet.

Forming Ground Truth. In practice, it is infeasible to compare coverage values to the ground truth (e.g., a test suite covers 70% of “diversity” which is inaccessible due to the infinite number of possible inputs in the wild). Besides, the coverage value is hardly interpretable (see discussion in Sec. VII). Consequently, the expected magnitude of the coverage induced by a test suite is unknown. To overcome these hurdles, we construct cases where the relative order of coverage values induced by different test suites is quantifiable (e.g., which test suite results in relatively larger coverage). The relative order forms a reasonable and measurable ground truth against which coverage values can be compared to determine their accuracy.

1) Diversity of Test Suites

Discriminative Models. To assess the diversity of test suites for discriminative models, we construct two datasets of different diversity using schemes denoted as $\times 1$ and $\times 10$. For the $\times 1$ scheme, we randomly choose 100 images from the test data and add white noise to produce dummy images (the total number of dummy images in $test_{\times 1}$ equals to the size of all test data). The white noise is truncated within $[-0.1, 0.1]$ to preserve the appearance of images. Given that text is discrete, it is infeasible to add white noise. We thus randomly shuffle words in the 100 selected sentences to produce dummy sentences. The $\times 10$ scheme follows the same procedure, but the number of the dummy data is 10 times of the test data.

Ground Truth (Table IV): The diversity of these datasets are ranked as $test_{\times 10} > test_{\times 1}$, because test contains over 10K diverse and meaningful samples in real-world datasets, whereas $test_{\times 10}$ and $test_{\times 1}$ are generated by (slightly) mutating 100 data samples.
Generative Models. Generative models generate images by sampling from a fixed distribution (i.e., the normal distribution). Since this procedure is conditioned on a given class label, we construct datasets of different diversity by following the Normal, $N/x$ and $C/x$ schemes (see Table V). For the $N/x$ scheme, we sample total $m$ inputs from the normal distribution and each class has the same number of inputs. For the $N/x$ scheme, we only keep $1/x$ of inputs from each class, whereas for the $C/x$ scheme, we only keep $1/x$ of classes. We set $m$ to $32,000$ and choose $x \in \{2, 10\}$.

Ground Truth (Table V): Since inputs conditioned on different classes are all sampled from normal distribution, behaviors of generative models are primarily decided by the class labels. Thus, the diversity of these datasets are ranked as: Normal $> N/2 > N/10 > C/2 > C/10$.

Result Overview. We show the increased coverage of different test suites compared with training data (since the DNN is already “tested” using its training data) in Table IV and Table V for discriminative image models and generative models. Due to the limited space, we report results of test (discriminative models in [1]), whose results are consistent with Table IV. All experiments are launched for five times to reduce randomness. The maximum deviations are around 0.01% and 1 for criteria in % and # units, respectively.

Scalability Issues of MDSC/DSC. In Table IV, we only evaluate MDSC using CIFAR10. MDSC requires storing class-conditional covariance matrices, whose space cost is $O(#\text{class} \times #\text{nunert}^2)$. Note that ImageNet has 1K classes and 1M training data (Table III). DSC takes over 12 hours to assess one scheme of ImageNet (criteria satisfying ) only take several minutes since it iterates over 1M neuron output traces for each test input. It’s thus infeasible to apply MDSC/DSC on models trained on ImageNet that is much larger than CIFAR10.

Impacts of Hyper-parameters. Previous criteria (except for NBC and SNAC) require extra hyper-parameters, which largely affect the results. For instance, all variants of CIFAR10 test data have nearly zero coverage when $T$ of NC is 0.25, whereas they reach a saturation coverage when $K$ of TKNP is 10 (i.e., patterns equals to #inputs) despite they are still evaluated on same models. Moreover, the hyper-parameter $T$ of CC is model specific, for instance, $T = 10$ is suitable for ResNet trained on CIFAR10, but is too small if the ResNet is trained on ImageNet — any input will form a new cluster. We thus increase $T = 100$

| Dataset | CIFAR10 | ImageNet | IMDB |
|---------|---------|----------|------|
| #Classes | 10 | 1,000 | 2 |
| #Data | 50,000/10,000 | 1,000,000/50,000 | 17,500/17,500 |

Tabled IV: Increased coverage on ResNet50 (discriminative model & image)†. Assessments matching ground truth are marked. Due to the limited space, we report results of VGG16 and MobileNetV2, which are consistent with this table, in [1].

| Model | #Neuron | #Layer | Remark |
|-------|---------|--------|--------|
| ResNet50 [17] | 5,027,957/5,560 | 54 | Non-spatial topology |
| VGG16 BN* [11] | 12,422,13/146 | 16 | Sequential topology |
| MobileNetV2 [20] | 17,000,018/356 | 51 | Mobile devices |
| BigGAN [8] | 6,404,682/1,120 | 10 | Generative model |
| LSM [19] | 32,769 | 129 | Text model |

*All image models have batch normalization (BN) layers.

| Dataset | CIFAR10 | ImageNet |
|---------|---------|----------|
| #Classes | 10 | 1,000 |
| #Data | 50,000/10,000 | 1,000,000/50,000 |

| Model | #Neuron | #Layer | Remark |
|-------|---------|--------|--------|
| ResNet50 | 5,027,957/5,560 | 54 | Non-spatial topology |
| VGG16 BN* | 12,422,13/146 | 16 | Sequential topology |
| MobileNetV2 | 17,000,018/356 | 51 | Mobile devices |
| BigGAN | 6,404,682/1,120 | 10 | Generative model |
| LSM | 32,769 | 129 | Text model |

*All image models have batch normalization (BN) layers.
TABLE VI: Coverage achieved by test data and AEs of CIFAR10†. Assessments matching ground truth are marked.

| Model  | NC  | KMNC | NBC | SNAC | TKNC | TKNP | LSN | DSC | MDSN | NLC  |
|--------|-----|------|-----|------|------|------|-----|-----|------|------|
| Test   | 0.75 | 100  | 0   | 0    | 0.36 | 1.03 | 55  | 44  | 146  | 73   |
| CW     | 0.90 | 3.94 | 2.73| 0.16 | 0.18 | 0.31 | 5   | 0   | 336  | 0.21 |
| PGD    | 1.43 | 0.35 | 0.35| 0.35 | 0.35 | 0.35 | 0.35| 0.35| 0.35 | 0.35 |
| V      | 2.49 | 0.78 | 0.78| 0.78 | 0.78 | 0.78 | 0.78| 0.78| 0.78 | 0.78 |
| Test   | 0.78 | 0.78 | 0.78| 0.78 | 0.78 | 0.78 | 0.78| 0.78| 0.78 | 0.78 |
| CW     | 0.20 | 5.03 | 7.08| 0.20 | 0.20 | 0.20 | 0.20| 0.20| 0.20 | 0.20 |
| PGD    | 1.69 | 12.64| 15.23|1.32 | 1.32 | 1.32 | 1.32| 1.32| 1.32 | 1.32 |
| M      | 1.57 | 4.81 | 0   | 0    | 0.21 | 0.21 | 0.21| 0.21| 0.21 | 0.21 |
| CW     | 0.02 | 1.77 | 1.62| 0.14 | 0.14 | 0.14 | 0.14| 0.14| 0.14 | 0.14 |
| PGD    | 0.36 | 0.71 | 0.39| 0.16 | 0.16 | 0.16 | 0.16| 0.16| 0.16 | 0.16 |

† R, V, M denote ResNet, VGG, MobileNet, respectively.

TABLE VII: Coverage achieved by AEs generated (via PGD) using test data of CIFAR10†. Assessments matching ground truth are marked.

| Model  | NC  | KMNC | NBC | SNAC | TKNC | TKNP | LSN | DSC | MDSN | NLC  |
|--------|-----|------|-----|------|------|------|-----|-----|------|------|
| ResNet | 0.41 | 0    | 0    | 0    | 0.17 | 0.44 | 0.40| 0.40| 0.40 | 0.40 |
| VGG    | 0.58 | 0    | 0    | 0    | 0.17 | 0.44 | 0.40| 0.40| 0.40 | 0.40 |
| MobileNet | 0.20 | 0    | 0    | 0    | 0.17 | 0.44 | 0.40| 0.40| 0.40 | 0.40 |

2) Fault-Revealing Capability of Test Suites

Setup. To evaluate the fault-revealing capabilities of test suites, we construct adversarial examples (AEs) using two adversarial attack algorithms, Carlini/Wagner (CW) [9] and Project Gradient Descent (PGD) [31]. CW and PGD are two representative types of algorithms for AE generation, namely, optimization-based and loss function-based. Perturbations added on AEs are different from white noise, as shown in Fig. 8, which generally highlight objects in images. AEs are generated using the training data (see “Clarification” below for results of AEs generated using test data), and all algorithms attacking the three models reach over 98% success rates. That is, more than 0.98 × (#training data) AEs can trigger erroneous behaviors. The incorrect predictions uniformly distribute across all classes.

Result Overview. As with the previous settings, we record and present the increased coverage of test data and AEs in Table VI. NC, TKNC, TKNP and CC have higher coverage for the test data than AEs, indicating that they are unable to assess the fault-revealing capability of a test suite. Harel-Canada et al. [16] also find that NC is not strongly correlated with defect detection, which is aligned with our results. Nevertheless, Table VI shows that NLC is strongly and positively correlated to the fault-revealing capability of a test suite.

Implementation Issues of LSC. MDSC has zero coverage on AEs generated using CW. Generally, perturbations added by CW are harder to identify [9]. This statement can also be illustrated by Table VI (nearly all coverage values of PGD are much higher than CW) and Fig. 8. Though LSC also gives zero coverage for CW AEs, it is seemingly due to some numerical issues. In particular, we observed some inf (infinite numbers) when LSC was computing the Gaussian kernel to measure the “surprise” of a test input. AE may induce a large neuron output, which consequently result in inf since exponential functions (f(x) = e^x) are used by the Gaussian kernel. DSC shows generally promising results for this evaluation, giving higher coverage values for AEs than normal test inputs.

Correlation to “Out-of-Bound” Neuron Outputs. KMNC has no response to AEs, but NBC and SNAC respond notably, because AEs primarily lead to out-of-range neuron outputs. NLC measures how divergent a layer output is and responds to the density change of layer outputs. Therefore, the out-of-range neuron outputs, which either induce a higher variance or introduce a low-density layer output, can be captured.

Clarification. Using training data to generate AEs helps alleviate side effects caused by the diversity of test data. To
TABLE VIII: Coverage achieved by adversarial-perturbed inputs (AP; not AE) generated using test data of CIFAR10*. Assessments matching ground truth are marked.

| Type         | Opacity  | Remark              | DSC: 10 | NLC: 10 |
|--------------|----------|---------------------|---------|---------|
| ResNet       | NBC: 0.05| SNAC: 0.1            | 0.20    | 0.09    |
| VGG          | NBC: 0.10| SNAC: 0.1            | 0.33    | 0.18    |
| MobileNet    | NBC: 0.05| SNAC: 0.1            | 0.26    | 0.21    |

Algorithm 1: Fuzzing algorithm.

```
1. Transformation Set: T; Fuzzing Seed: S;
2. Tested DNN: D; Criterion: C; num ← 50;
3. while not terminate() do
   4. i = sample(S);
   5. for i = 0 to num by 1 do
      6. t = sample(T); i ← i(t);
      7. if is_valid(i, S) and coverage_inc(C, D) then
         8. S.add(i); update_coverage(C);
         9. break;
```

Fig. 9: Stylizing and blurring.

(Original image) (Stylized) (Blurred)
TABLE XI: Fuzzing results. F, O, C, E denote triggered faults, fuzzing outputs, classes, scaled entropy, respectively.

| CIFAR10 | ImageNet |
|---------|----------|
| #/ID   | IS/FID  | RC | E   |
| GMna   | 2,980/10,000 | 9,08 185 | 8.73 | 9,08 145 | 9.95 | 0.82 |
| NC     | 1,260/1,100 | 1,45 | 175 | 9.67 | 1,300/1,100 | 9.85 | 0.93 |
| RSNC   | 60 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| NBC    | 8/10 | 1,39 | 45 | 0.97 | 1,05 167 | 4.36 |
| SNAC   | 38 8 | 7,04 46 | 9.01 | 0.10 | 2,62 172 | 4.97 |
| TKNP   | 5,140/1,150 | 9,30 170 | 7.15 | 9,10 170 | 9.30 | 0.83 |
| TKNC   | 5,140/1,150 | 9,30 170 | 7.15 | 9,10 170 | 9.30 | 0.83 |
| CC     | 39/17 | 2,55 358 | 9.02 | 0.02 | 1,10 127 | 9.08 | 0.86 |
| LSC    | 7,50 250 | 7.98 350 | 9.02 | 0.02 | 1,10 127 | 9.08 | 0.86 |
| DSC    | 7,50 250 | 7.98 350 | 9.02 | 0.02 | 1,10 127 | 9.08 | 0.86 |
| NBC    | 2,10 121 | 7.50 250 | 9.02 | 0.02 | 1,10 127 | 9.08 | 0.86 |
| SNAC   | 2,10 121 | 7.50 250 | 9.02 | 0.02 | 1,10 127 | 9.08 | 0.86 |

*Higher E is better if two #C are equal, otherwise, higher #C is better.

TABLE XII: Time cost for 10K CIFAR10 inputs on ResNet.

| C        | NM | KMNC | NBC | SNAC | TKNC | TKNP | CC | LSC | DSC | NBC | SNAC | TKNP/KMNC/DSC
|----------|----|------|-----|------|------|------|----|-----|-----|-----|------|------------------|
| Time     | 3.2 | 2.45 | 2.45 | 2.66 | 3.15 | 3.06 | 2.77 | 3.15 | 2.77 | 3.15 | 3.06 | 2.77 |

10-2 \int C/|C|

Images [16]. We measure the naturalness of mutated inputs under each criterion; a good criterion should lead to mutations that mostly preserve the naturalness. We use Inception Score (IS) [39] and Fréchet Inception Distance (FID) [18], which are widely used in the AI and SE community to assess the naturalness of images, as the metrics. A higher IS score is better, whereas a lower FID score indicates better naturalness. The evaluation results are also listed in Table XI.

Random mutation’s IS and FID scores are the baselines. Overall, mutated images guided by NLC have the best scores for both IS and FID. Mutated images guided by SCs, NBC, SNAC have relatively low naturalness. SCs prefer inputs that are distinct to training data (also pointed out in Sec. VI-A1). Given the high comprehensiveness of real-world datasets, SCs may likely push to produce recognizable images. NBC and SNAC denote a neuron as activated if it has an out-of-range outputs. Thus, its guided mutations tend to largely modify pixel values. We find that the mutated images become less recognizable, e.g., a region in an image is covered by a black block; see [1] for more examples. Overall, since neurons jointly process features in an image, mutations guided by NLC, which aims to cover different neuron entanglements, are holistically “bounded” by features.

Diversity of Erroneous Behaviors. A higher diversity of erroneous behaviors indicates uncovering a practically larger vulnerability surface of DNN models. Overall, a collection of fault-triggering images are regarded as more diverse if they cover more classes. In case that covered classes is equal, we further use the scaled entropy, $-\frac{1}{C} \sum C_i \log p_i$, where $p_i$ is the ratio of incorrect outputs predicted as class $c$ and $|C|$ is the classes, to assess the diversity: a higher entropy is better. The results are also presented in Table XI. It is seen that fault-triggering images generated by using NLC as the guidance cover the most classes. We thus envision equipping fuzz testing frameworks with NLC to likely identify more and diverse defects from real-world critical DNN systems.

VII. DISCUSSION

Time/Space Cost of NLC. Table XII compares the time cost of NLC with previous criteria. NLC is slower than NC/NBC/SNAC/TKNC due to covariance computation. It is slightly faster than KMNC/MDSC and much faster than TKNP/CC/LSC/DSC.

Despite both MDSC [23] and NLC use covariance matrix, NLC requires less memory space. Let $m_i$ be the neurons in the $i$-th DNN layer and $K$ be the classes in training data, NLC takes $\sum m_i^2$ space, while MDSC takes $K \times (\sum m_i)^2$. To interpret the comparison: first, $K$ is large in real-world datasets ($K = 1000$ in ImageNet), hence MDSC may be hardly used on ImageNet-trained DNNs but NLC can. Second, when $K = 1$, NLC still takes less space and saves more space for deeper DNNs.

Interpretability and Usage. We view that DNN coverage criteria (including NLC and prior works) may share common limitation of shallow interpretability on their outcomes; For instance, as noted in Sec. III-B, NC simplifies the neuron output (a continuous value) to a binary state (activated vs. unactivated). Hence, given a case of 100% NC coverage, we are concerned that different neuron output values and the combination states of neurons/layers are far from fully examined. Worse yet, once it hits 100% (merely CIFAR10 training data can have 100% coverage on ResNet using ⟨NC, T = 0.2⟩), further tests cannot be guided by NC. That is, “saturating” state of DNN criteria should not be interpreted as revealing all DNN behaviors. Similarly, zero coverage reported by some criteria does not imply that DNN behaviors are never explored. In Table IV, when using the same inputs (CIFAR10), NBC has zero coverage, whereas ⟨TKNP, K = 10⟩ is saturated (increases 10,000 with 10,000 inputs).

High-level speaking, DNN coverage is not simply analogous to software coverage, where each statement is either “covered” or not. Outputs of DNN coverage criteria depend on how those criteria, not merely the DNNs, are designed/implemented. Fundamentally, neuron outputs are continuous, and neurons in DNN layers collaborate to comprehend the semantics of high-dimensional inputs (e.g., images). Explaining DNN decisions is still an open problem. Thus, DNN coverage is not as clear as traditional software coverage, where we simply “count” if a statement/path/basic block is covered or not.

In sum, aligned with recent criteria like CC [32], we do not strive to establish a “maximal value” for NLC (since misprediction cannot be eliminated currently). To use NLC, developers should try their best to maximize NLC values with test inputs. Also, it is infeasible to universally decide the number of inputs required to maximize NLC, as this number depends on the DNN structures, weights, and training data.

VIII. CONCLUSION

Given observations that DNNs approximate distributions, we propose eight design requirements for DNN testing criteria. We design NLC, a criterion that fulfills all requirements. NLC considers the continuity, correlation, distribution, and density of layer outputs in a DNN. It also enables various optimizations. We show that NLC can facilitate test input assessment and serve as guidance of fuzz testing with better performance.
[46] Xiaofei Xie, Tianlin Li, Jian Wang, Lei Ma, Qing Guo, Felix Juefei-Xu, and Yang Liu. Npc: Neuron path coverage via characterizing decision logic of deep neural networks. *ACM Transactions on Software Engineering and Methodology*, 2022.

[47] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Hongxu Chen, Minhui Xue, Bo Li, Yang Liu, Jianjun Zhao, Jianxiong Yin, and Simon See. Coverage-guided fuzzing for deep neural networks. *arXiv preprint arXiv:1809.01266*, 2018.

[48] Guorong Xuan, Wei Zhang, and Peiqi Chai. Em algorithms of gaussian mixture model and hidden markov model. In *Proceedings 2001 International Conference on Image Processing (Cat. No. 01CH37205)*, volume 1, pages 145–148. IEEE, 2001.

[49] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[50] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. DeepRoad: GAN-based Metamorphic Testing and Input Validation Framework for Autonomous Driving Systems. In *ASE*, 2018.