Black-Box Testing of Deep Neural Networks

Taejoon Byun
University of Minnesota
Minneapolis, MN, United States
taejoon@umn.edu

Sanjai Rayadurgam
University of Minnesota
Minneapolis, MN, United States
rsanjai@umn.edu

Mats P.E. Heimdahl
University of Minnesota
Minneapolis, MN, United States
heimdahl@umn.edu

***Abstract*—Several test adequacy criteria have been developed for quantifying the the coverage of deep neural networks (DNNs) achieved by a test suite. Being dependent on the structure of the DNN, these can be costly to measure and use, especially given the highly iterative nature of the model training workflow. Further, testing provides higher overall assurance when such implementation dependent measures are used along with implementation independent ones. In this paper, we rigorously define a new black-box coverage criterion that is independent of the DNN model under test. We further describe a few desirable properties and associated evaluation metrics for assessing test coverage criteria and use those to empirically compare and contrast the black-box criterion with several DNN structural coverage criteria. Results indicate that the black-box criterion has comparable effectiveness and provides benefits that complement white-box criteria. The results also reveal a few weaknesses of coverage criteria for DNNs.**

I. Introduction

Software testing approaches are traditionally categorized as *white-box*—techniques that utilize the internal structure of the program under test—or *black-box*—techniques that utilize artifacts of higher abstraction. For assessing test adequacy, white-box criteria measure the coverage over the structure of the program, whereas black-box criteria measure coverage of high-level artifacts such as requirements, specification, or input domain [1], [2]. While white-box criteria have strengths, including being effective at finding implementation defects, when relied on exclusively, they entail the risk of missing the forest for the trees—e.g., a missing functionality may escape undetected. Black-box testing provides a complementary top-down perspective, needed to overcome those short-comings. Both are thus necessary, and in practice, used in tandem.

Testing of DNNs is challenging due to many reasons—there are no well-accepted ways to derive test cases, and no established ways of measuring how well a test suite has exercised the DNN's behavior. While several metrics have been proposed to assess the coverage of a DNN under test, those are broadly along the lines of *white-box* coverage criteria: they quantify the coverage of the internal DNN structure—e.g., by counting the number of neurons that have been activated by a test suite. Though potentially useful, the rationale for using the DNN structure as a basis for coverage measurement is not apparent. E.g., what does it *really mean* when a neuron is, or is not, covered, and what should one do about it. Moreover, these criteria have the same limitations as traditional white-box criteria in their dependence on the program under test, with the impact further exacerbated due to the highly iterative nature of the machine-learning (ML) development workflow. Small changes to the DNN layers or weights, can have large consequences for the measured coverage, invalidating any analysis done with prior coverage information. A black-box coverage criterion expressible in terms of the input domain would avoid such issues.

In this paper we rigorously define a black-box coverage criterion called *Manifold Combination Coverage* (MCC) that measures the test adequacy on a high-level artifact called a *manifold*, which is analogous to an input domain model in black-box testing. This artifact is created using *manifold learning*, which learns a compact subspace within the input space, along with an encoder and a decoder function that maps the input space to and from the manifold. Similar to black-box testing in a traditional sense, the use of MCC has the benefit of being independent of the model under test, is inexpensive to measure, scales well, and is amenable to interpretation.

While preparing to empirically assess MCC, we noticed that a standardized approach to evaluate and compare ML test adequacy criteria, taking into account how they are to be used in practice, is missing. Therefore, we first clarify the usages of coverage criteria in the ML workflow and define three metrics for empirically comparing the criteria as indicators of test suite quality— *failure-revealing effectiveness* (how well a test suite triggers failures), *semantic balance* (how well a test suite resembles the expected input distribution), and *retraining efficacy* (how well a test suite *fixes* bugs via retraining). Using these metrics, we experimentally compare MCC and several white-box criteria on nine realistic models trained for MNIST [3], CIFAR-10 [4], and Udacity [5]. In addition, also used a previously defined test prioritization technique [6] to learn how they influence the effectiveness of coverage criteria. The results show that 1) MCC is more effective than white-box criteria in constructing a semantically balanced test suite without sacrificing failure-revealing effectiveness much, 2) the effectiveness of coverage criteria is highly sensitive to the dataset, DNN, and coverage configurations, and 3) prioritization has a pronounced positive effect on the effectiveness of test suites constructed with coverage criteria.

In short, the contributions are 1) a rigorous definition and evaluation of a black-box coverage criterion for testing DNNs, 2) metrics and method for objectively evaluating DNN coverage criteria, and 3) an empirical comparison of the effectiveness of DNN coverage criteria and prioritization measures in the context of testing DNNs for generalization [7].

II. Testing Deep Neural Networks

ML testing is defined as any activity designed to reveal imperfections in the ML system under test that causes a *discordance* between existing and desired conditions [8]. These desired conditions are called *properties*, and the manifested discordance are called *failures*. The latent cause of a failure is called a *fault*, which can be in the training data, the model definition, or the learning framework [9]. Compared to hand-written software, a fault in an ML system can be more subtle and difficult to identify, as the decision logic is *learned* using a learning algorithm, rather than being coded explicitly. The eventual goal of ML testing is to identify and fix these faults, but an immediate prerequisite is to trigger as many diverse failures as the budget allows. In this paper, we discuss ML testing in the context of revealing failures in DNNs.

A. Testing in Machine Learning Workflow

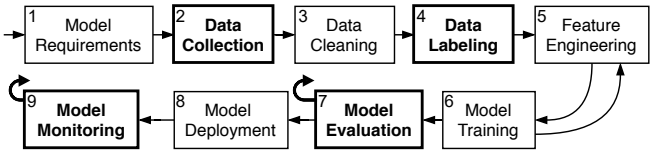

Fig. 1: The nine stages of a machine learning workflow [10]

ML testing cannot be discussed properly without considering the context in which it is performed, especially since the workflow of developing an ML system is somewhat different from a traditional software development lifecycle [11]. According to a recent ML workflow model suggested by Amershi *et al*. [10] (Figure 1), the most prominent differences are frequent iterations and feedback loops. These loops exist in many layers. For instance, stages 1 through 7 may be performed on a daily basis with updated dataset each time. Stages 3 through 7 may be iterated multiple times in a sitting to determine which data needs to be cleaned. Stages 5 through 7 may be iterated for feature engineering, and stages 6 and 7 may be iterated hundreds of times, often by an algorithm, for tuning the hyper-parameters.

During training, the model is typically tested with a *validation set*, which is a dataset that is set aside from the training dataset for an unbiased evaluation of the model's performance during training and hyper-parameter tuning. To select one final model among candidates, or to evaluate the *real-world* performance of the selected model, another independent dataset called *test set* is used. In this paper, we only consider testing in the latter context (in stage 7) where one attempts to discover real failures that may occur once the ML system is deployed.

B. Testing Properties

While the immediate goal of testing is to find many failures, not every failure points to the same issue. For example, it is one thing for a system to confuse a mule with a donkey (they look very similar), and quite another to confuse a donkey with a cat because the image was adversarially perturbed [12]. The former case is often called a generalization error, whereas the latter is called an adversarial robustness issue. These categories of desired behaviors, which also define the class of failures, are called generic properties. Testing strategy depends heavily on what property one aims to test for, although this aspect is rarely discussed explicitly in the literature. We discuss below, a few generic properties most relevant to testing of DNNs for computer vision tasks, to place our approach in proper context.

Robustness is a property of an ML system that indicates the "degree to which it can function correctly in the presence of invalid inputs or stressful environmental conditions" [13].

Adversarial Robustness: This is a sub-category of robustness in which failure causing inputs are constrained to be indistinguishable from benign ones [14]. For a neural network f, *δ-local adversarial robustness* is defined as $\forall x' \cdot ||x - x'|| \leq \delta \implies f(x) = f(x')$, meaning that a small adversarial perturbation of size δ or smaller shall not change the output of the model. Global-adversarial-robustness is defined similarly for every pair of input (x_1, x_2) [14]. Testing for adversarial robustness often means revealing the absence of adversarial robustness by creating or finding adversarial examples.

Corruption Robustness: This, in contrast to adversarial robustness, quantifies the robustness under corruptions that may occur in a normal environment, such as change in brightness or sensory noise. Unless the objective of testing is to find security vulnerabilities, corruption robustness more closely captures the real-world robustness requirement for the MUT. When f is the MUT, $\mathcal{D}_{gt}$ is the ground-truth distribution of the data, and $\mathcal{C}$ is the distribution of corruptions in the real world, corruption robustness [15] is defined as $\mathbb{E}_{c\sim\mathcal{C}}[\mathbb{E}_{x\sim\mathcal{D}}\mathbb{I}[f(c(x)) = f(x)]]$, where x is an input and $\mathbb{I}$ is the indicator function. In other words, it is the expected performance of the model f measured by metric $\mathbb{I}$ under expected set of corruptions $\mathcal{C}$. The goal of testing for corruption robustness is to find $c \in C$ and $x \in D$ such that $f(c(x)) \neq f(x)$.

Generalization: Although corruption robustness expresses a more general requirement than adversarial robustness, the operational domain of DNN is not limited to enumerable corruptions. In fact, the goal of machine learning can be framed as training a model f that *generalizes* to any unseen data drawn from a hypothetical ground-truth population $\mathcal{D}_{gt}$, instead of simply memorizing the training data [16]. Thus, the generalization capability of a model needs to be tested just like testing the functionality of any software. The generalization of the model is quantified as: $\gamma_{gt} : \mathbb{E}_{(x,y)\sim\mathcal{D}_{gt}}\mathbb{I}[f(x) = y]$. However, since the ground truth distribution is unknown, the performance of a trained model is instead measured on a test suite D_{test} as follows [7]: $\gamma_{test} : (1/|D_{test}|) \sum_{(x,y)\in D_{test}} \mathbb{I}[f(x) = y]$. If f generalizes, the performance of the model measured on a sufficiently large test dataset D_{test} drawn from $\mathcal{D}_{gt}$ shall match the true performance measured on $\mathcal{D}_{gt}$. The goal of testing for generalization can be (1) to find $(x, y) \in D \cdot f(x) \neq y$, and (2) to find a test suite $\mathcal{D}_{test} \sim \mathcal{D}_{gt}$ that reveals discrepancy between the desired γ_{gt} and the actual γ_{test}. The first goal is achieved by finding what is called natural adversarial examples [17], [18], or test cases that are found in the real world that trigger failures. The

second goal is harder to achieve, as it also requires that the distance between the two distributions $\mathcal{D}_{test}$ and $\mathcal{D}_{gt}$ is small.

In many recent ML testing papers, the distinction among these properties were not made clear. Adversarial robustness was considered to be *the goal* of testing in many works. But it is only one property of the model in a specific kind of (adversarial) environment, and whether it correlates to the actual performance of the model in a non-adversarial environment is unknown. We believe that testing for generalization differs from testing for adversarial robustness in the same manner as functional testing differs from penetrating testing. Each has a distinct purpose and must be tested for separately. In this work, we discuss testing only in the context of generalization. We view corruption robustness as special case of generalization in that real-world data distribution will include corruptions at their natural frequency of occurrence.

III. White-Box Coverage Criteria for DNNs

Exhaustive testing is almost always infeasible. Practitioners make compromises to achieve a maximum effectiveness—*i.e.*, find many faults—while staying within budget. Test adequacy criteria provide a practical guidance in achieving this goal through a quantitative measure of test adequacy given a test suite. When used during test suite construction, coverage can help avoid duplicate test-cases, and can help ensure some form of completeness so that a whole range of system behavior can be tested. As code-based criteria are not applicable to DNNs, recent works have proposed new criteria designed for testing DNNs. However, given the ML workflow, there are limitations to these criteria that are often overlooked. This section revisits the usage scenarios and the utility of DNN-specific coverage criteria and further discusses their limitations.

A. Use-Cases of Coverage Criteria

In traditional software development, test adequacy criteria are used as proxy measures of test completeness [19] in various stages of verification and validation (V&V). An ML coverage criterion can also be used in various stages of the ML workflow (Figure 1), and its primary goal is also a high correlation with the failure-revealing capability of tests. In the ML workflow, however, its use-case can extend further beyond model evaluation to any stage where a judgement on the adequacy of a dataset becomes useful. These stages include test data collection, test-data labeling, model evaluation, and monitoring (highlighted in Figure 1). The utility of a coverage criterion can also extend beyond finding more faults. For instance, one may want to use a test suite to *fix* the model through retraining. In other cases, one may want to obtain a semantically *balanced* test suite such that it can evaluate the performance of the model objectively.

B. White-Box Testing of DNNs

Traditional white-box criteria measure the coverage over the internal structure of the program under test. The rationale is that for a test suite to find a fault, it is necessary for it to execute the faulty logic inside the program first. Similar rationale and expectations seem to carry over to DNN coverage criteria, albeit fault may be attributed to any structural fragment of a DNN.

Since the first proposal of Neuron Coverage (NC) [20], increasingly more sophisticated structural criteria have been proposed [21]–[30]. These criteria base their measurement of test completeness on the MUT, analogous to measuring structural coverage on traditional programs. We briefly explain some of the most popular criteria. • NC requires a test suite to *activate* all the neurons inside a DNN, where the activation is determined by a pre-configured threshold. • k-multisection neuron coverage (KMNC) [21] is a granular extension of NC that partitions the range of neuron's output into k sections, and requires a test suite to cover each of them. • Neuron boundary coverage (NBC) [21] requires a test suite to exercise the neurons beyond the range of values known from the training data, with the intention of exercising corner-case behaviors of a DNN. • Surprise adequacy (SA) [22] takes the activation traces from the DNN—thus qualifies as a white-box criterion—and interprets them as representations of input data. The relative surprise of an input is measured by its relative proximity to inter-class and intra-class data, where the distance is measured between the activation traces. The coverage is defined by computing the range of surprise values, dividing it into n sections, and by requiring a test suite to cover the whole range of surprise values.

C. Limitations of White-Box Testing of DNNs

Due to the difference between the ML workflow and a traditional software development life-cycle, white-box testing of DNNs has several shortcomings, such as its model-dependence, high cost of measurement, and lack of interpretability. We further elaborate the two major shortcomings—model-dependence and high measurement cost.

Model Dependence: The ML workflow is highly iterative (Figure 1). It is necessary to experiment with different configurations to obtain a good model. Unlike in a traditional software development life-cycle where the program evolves incrementally, the structure of a DNN can change completely by changing only a few settings among many moving parts—such as training data, the DNN architecture, and hyper-parameters. This nature of ML development makes white-box testing less suitable. For example, suppose a scenario of testing two different DNNs with NC where the first model dominantly uses *sigmoid* for activation whereas the second model uses *tanh*. For the first model, a near 100% neuron coverage will be achieved with a single input, as sigmoid output is always greater than 0. The coverage will be much lower for the second model, since the range of $tanh$ output is in $[-1, 1]$. This difference in coverage does not indicate the difference in the relative thoroughness of testing nor the quality of the model. It means that even for two models that are equivalent in performance, when tested with the same test suite, the reported adequacy scores can be dramatically different, while this difference is not necessarily associated with the adequacy of the testing.

High Measurement Cost: The time and space complexity of many popular DNN coverage criteria are linear in the number of neurons inside a DNN. The size of DNNs used in practical applications can be enormous, ranging up to a few tens of million parameters for computer vision models [31]. As the complexity is also linear in the number of test cases and in the number of models to test, the overall cost quickly becomes prohibitively expensive. For example, the coverage vector for *a single test run* on a 100M-parameter model consumes $10^8/(8\times 2^{20}) = 11.9$ MB of memory space, accompanied with the I/O overhead of copying the DNN intermediary output from GPU to RAM. In fact, the experiment performed in DeepGauge [21] for evaluating various white-box criteria had to employ a compute cluster where each node is equipped with 196GB-RAM—a non-trivial cost for coverage measurement.

IV. Black-Box Coverage Criterion

Black-box testing is a testing method that does not require any information about the internals of the software under test. When measuring the test adequacy, black-box techniques rely on artifacts that are in higher level of abstraction, such as input description, design model, specification, or requirement [2], [32]. Consequently, a black-box adequacy criterion for ML is independent of the MUT, and can be applied to any data-centric software. By being model-independent, the aforementioned shortcomings of white-box criteria are easily resolved, and a black-box criterion can scale even to complex software system that is composed of multiple ML models. We introduce such a black-box criterion in the sequel.

A. Manifold Combination Coverage

Manifold Combination Coverage (MCC), a concept proposed by Byun and Rayadurgam [33] and formalized below in Def. 4.1, is analogous to combinatorial coverage for traditional black-box testing [34]. The key idea is to assess test adequacy based on the semantics of the input instead of the structure of the MUT. The challenge for DNN input-space coverage, however, is that the input space is often highly unstructured and high-dimensional. For instance, it would be impractical to apply combinatorial testing on a 256x256 image space. A pixel-level coverage is too granular to infer any meaning, and combination over 256^2 pixels is infeasible. How can we effectively summarize the high-dimensional input space while preserving the information relevant for the MUT?

We address this challenge with an unsupervised learning technique called manifold (representation) learning [35]. Mathematically, *manifold* is a topological space that locally resembles an Euclidean space near each point, and *manifold learning* rests on an underlying principle called the *manifold hypothesis* which posits that real-world high-dimensional data—such as images—lie within an embedded low-dimensional subspace called manifold. With manifold learning, high-dimensional input space can be compressed down to a lower dimensional manifold space, and as a result, the data can be summarized with a much smaller number of surrogate variables that encode semantic features of the input

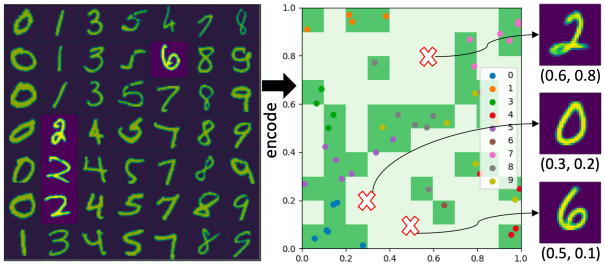

Fig. 2: Coverage analysis on 2D Manifold Space.

data. Once we have the capability of projecting any test data onto the manifold space, we can measure coverage on this space to gauge the thoroughness of testing. With its inverse function that can map the encoded test cases back to the input space, uncovered coverage obligations can be interpreted by analyzing concrete samples generated from them.

The gist of our approach is to map the high-dimensional input data to $\mathbb{R}^d$ for a small enough d using an encoding technique and to partition each dimension into k contiguous ranges such that the partitioning is *uniform* with respect to the sample density of a reference set (e.g. the training data set for the MUT), i.e., an arbitrarily chosen element of the set has equal probability of falling into any of the k partitions. MCC of a test suite can then be taken as the fraction of the k^d possible combinations of the partitions that the test data in the suite map to. With the two tunable parameters k and d, we define (k, d)-Manifold Combination Coverage as follows.

Definition 4.1 ((k, d)-Manifold Combination Coverage): The **k-section d-dimesional manifold combination coverage** of an input data domain $\mathbb{X}$ with respect to an encoding function $q : \mathbb{X} \to \mathbb{R}^d$ and a binning function $\rho_k^d : \mathbb{R}^d \to \mathbb{Z}_k^d$, is defined as a real-valued function $\mu : 2^{\mathbb{X}} \to [0, 1]$ given by $\mu(A) = \left|\left\{\rho_k^d(q(x)) \mid x \in A\right\}\right| / k^d$ for any $A \subseteq \mathbb{X}$.

$\mathbb{Z}_k$ denotes the set $\{0, ..., k-1\}$. The function $q(\cdot)$ encodes input data to the manifold space and the function ρ_k^d maps the low-dimensional data with the unique combination of the partitions of the dimensions that it falls into. The binning function ρ_k^d we used in this paper divides each dimension into k contiguous ranges containing equal training sample density per bin in each dimension. If the implicit semantic features of the dataset can be learnt as a manifold, and if k partitions are seen as distinct possibilities for each of those features, then manifold combination coverage requires inclusion of all possible interactions of those features, analogous to combinatorial testing of traditional software which can expose interaction faults. Since semantic closeness is preserved as Euclidean distance in the manifold space, MCC also ensures that duplicate samples do not count for coverage.

Figure 2 illustrates the concept with $(10, 2)$-MCC for testing a model trained with MNIST dataset [3], where $d = 2$ was chosen for a 2-D visualization. Among 49 test cases presented on the left, 45 cases (shaded) were encoded to the manifold space and covered $34/10^2$ coverage obligations (highlighted in green). The number of covered obligations is lower than the number of selected test cases, because some coverage

obligations are satisfied by more than two semantically close test cases. Test cases with label 2 and 6 were not selected intentionally. But by analyzing the uncovered obligations, and by using the inverse function that maps the code back to the input space, we can *see* that some of the uncovered obligations are due to missing test cases of label 2 and 6. Also, by analyzing an uncovered obligation near the area covered by 0s, a 0 with a distinct style could be generated, informing the tester what kind of test input is missing in the given test suite.

B. Manifold Learning with Variational Autoencoder (VAE)

We chose VAE among many manifold learning approaches. It is a latent-variable generative model that is capable of producing outputs similar to inputs by determining a latent-variable space Z and an associated probability density function $P(z)$. Its goal is to ensure that for each datapoint in a given dataset $x \in X$, there is at least one valuation of the latent variables $z \in Z$ which causes the model to generate $\hat{x}$ that is very similar to x. The first part that maps x to z as $z = q(x)$ is called encoder, and the second part that maps z back to x as $\hat{x} = p(z)$ is called decoder. This pair of q and p is instantiated with DNNs. We refer to a tutorial for more details [36].

One of its distinctive features that matters the most for coverage measurement is that we can control the distribution of samples in the manifold space. VAE requires a prior distribution to be specified, and training minimizes the divergence between the posterior distribution of z and the specified prior distribution. Multivariate normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I}_d)$ is typically chosen as a prior. However, this does not ensure that the posterior distribution will match the prior distribution perfectly, and furthermore presents non-zero covariance among dimensions. Our binning function defines the ranges based on the posterior distribution to account for this mismatch.

As a practical consideration of configuring a VAE, the size of the latent dimension d has an impact on the precision of coverage measurement. Although in theory any d can encode an arbitrarily complex dataset, a higher d allows for a more capacious latent representation in practice, and offers a more fine-grained differentiation. However, the benefit of a higher d also comes with the curse of dimensionality [37] and combinatorial explosion thereafter. A practical upper-bound can be determined through a binary search while minimizing the VAE loss. This d can then be reduced to fit the testing budget, at the cost of sensitivity to finer-grained features.

V. Effectiveness of Coverage Criteria

While pursuing the quest of evaluating the effectiveness of MCC against other white-box criteria, we discovered that a standardized method is missing in the ML testing literature for an empirical evaluation of coverage criteria. As most existing works on DNN coverage evaluated their effectiveness only in the adversarial testing context, the claimed effectiveness may not extend to other contexts such as when testing for generalization. Moreover, the notion of effectiveness itself has not been well established. So, we first define the metrics with which to empirically evaluate ML coverage criteria.

A. Evaluation Criteria

Coverage score is often interpreted as a measure of failure-revealing capability of a test suite, and this intuition is named *Strong Coverage Hypothesis (SCH)* [38], which says "there is at least a moderate statistical correlation between the level of coverage a suite achieves and its level of fault detection." SCH can be tested empirically by repeatedly measuring the correlation for a statistically significant number of test suites. If SCH holds for a criterion c, test suites T_1 and T_2 that achieve different coverage scores, with $c_{T_1} > c_{T_2}$, can be taken to mean that T_1 finds more faults than T_2. SCH was shown to hold weakly for real-world programs [38]–[40], but has not been evaluated for DNN coverage criteria.

Unlike in traditional software, it is hard to discuss the effectiveness in terms of fault finding for DNNs since a fault in DNN is a product of multiple factors that influence the trained model. Moreover, failure-revealing effectiveness itself is not the only goal because, when the purpose of testing is also to evaluate the performance of the model objectively, the semantic balance of the test suite—a low divergence between $\mathcal{D}_{test}$ and $\mathcal{D}_{gt}$—also matters. Hence, we capture the efficacy of coverage criteria from three different aspects—failure-revealing effectiveness, semantic balance, and retraining efficacy. These metrics are quantified over a test suite. The effectiveness of coverage criteria is measured indirectly through test suites created with coverage criteria under investigation.

Failure-Revealing Effectiveness: This metric measures the effectiveness of a test suite in picking out test cases that reveal failures. Failure-revealing effectiveness of a test suite T is quantified as the percentage of test cases that reveal generalization failures: $\{(x, y) \in T | f(x) \neq y\} \,/\, |T|$.

Semantic Balance: This metric measures how a test suite matches $\mathcal{D}_{gt}$, and in practice, we consider $\mathcal{D}_{train}$ as approximating $\mathcal{D}_{gt}$. Since comparing the distributions between high-dimensional datasets T and $\mathcal{D}_{gt}$ is infeasible, we use available semantic features of a test dataset for comparison. Semantic balance is defined as 1.0 minus the Jensen-Shannon divergence between the two datasets as: $1.0 - JSD(P_{feat}(T)||P_{feat}(\mathcal{D}_{gt}))$, which evaluates to 1.0 when the two distributions are identical and converges to 0.0 when the two diverge. In the experiments, we measure the distribution with respect to the class label, which is the only available feature for our datasets. Real-valued labels are discretized.

Retraining Efficacy: This metric measures the performance of a model after retraining with T. Failure-revealing inputs are like evidences of faults, and multiple evidences may stem from the same latent cause. As the number of faults cannot be measured directly, retraining efficacy measures how much of the errors caused by a lack of training data can be mitigated by adding the selected data to the training set. T_1 is said to achieve a higher retraining efficacy than T_2 if $D_{train} \cup T_1$ yields a model of higher performance after retraining than when trained with $D_{train} \cup T_2$, given that every other variable such as epochs and learning rate are fixed.

B. Test Prioritization and Coverage

In terms of failure-revealing effectiveness, there is another technique known to be effective—test prioritization [6], [41]. In one of the studies [6], it was shown that the top 20% of the prioritized test inputs included more than 80% of the failure-revealing inputs in the pool. This result indicates that test prioritization can affect the effectiveness of test suites when applied together with coverage criteria for coverage-directed test creation or selection. This is applicable only when there are more test cases available in the pool than what the testing budget allows for execution, but such a scenario is a practical case since it is relatively easy to collect large amount of unlabeled data, while the testing budget—the cost of labeling, running, and analyzing the result—is finite.

In the experiment, we consider prioritization as an independent variable when comparing the coverage criteria. We adopted two prioritization measures [6]—softmax cross-entropy for classification models, and Bayesian uncertainty for regression models. Softmax cross entropy quantifies how *certain* the classifier is for its prediction, and for a test input x, it is defined as $-\sum_{c=1}^{C} p_c \log p_c$, where $p = f(x)$ is the probability vector produced by the classifier f, p_c denotes the c-th element of p, and C is the total number classes. Bayesian uncertainty is defined as the variance of the output from a Bayesian neural network, but it can be extended to non-Bayesian neural network with dropout layers since dropout layers can approximate a Bayesian neural network [42]. The predictive variance is computed through a finite number of Monte-Carlo simulations as $Var(y) \approx \frac{1}{T}\sum_{t=1}^{T} f(x)^T f(x) - E(y)^T E(y)$ with T samples and the predicted mean $E(y) \approx \frac{1}{T}\sum_{t=1}^{T} f(x)$ [43].

VI. Experiment

In the experiment, we compare the effectiveness of MCC against white-box criteria. We formulate the research questions based on the metrics defined in Section V-A:

- **RQ1**: Is manifold coverage as effective as white-box criteria in terms of *failure-finding* effectiveness?
- **RQ2**: Is manifold coverage as effective as white-box criteria in creating *semantically balanced* test suites?
- **RQ3**: Is manifold coverage as effective as white-box criteria in creating a dataset for *retraining* the model?

For each research question, we compare the effectiveness with and without the prioritization separately.

A. Datasets and Models Under Test

The experiments are performed on three different computer vision tasks—two classification tasks and one regression task. A classification model outputs a Bernoulli distribution for the classes it predicts, and the class with the highest predicted probability is considered as the output of the model. A regression model outputs a set of real values.

The first task is a ten-class hand-written digit classification, for which we trained three models with MNIST dataset [3] consisting of 60,000 black-and-white images of 28 x 28 pixels. The second task is CIFAR-10 image classification [4]—the training data consists of 50,000 color images of size $32\times32\times3$. We trained three CIFAR-10 models of the same architectures. The third task is Udacity self-driving car dataset [5] which is a regression task for predicting the steering angle based on the dashboard camera images. We trained the same set of models, with 27,046 train data of size $96 \times 96 \times 3$. These models are implemented in and trained with TensorFlow 2.3 [44], and the implementations are adapted from the code written by Li [45]. The details of each model is described in Table I. Their train, validation, and test performance measures are presented as classification accuracy for MNIST/CIFAR-10 and mean absolute error (MAE) for Udacity.

TABLE I: Models under test

Task	Architecture	# Params	Train	Val.	Test
MNIST	LeNet [46]	44k	98.53%	98.86%	91.81%
	Network-in-Network [47]	957k	99.13%	99.11%	90.17%
	ResNet-32 [48]	468k	99.65%	99.41%	90.87%
CIFAR-10	LeNet	62k	72.43%	72.43%	68.87%
	Network-in-Network	967k	93.00%	87.90%	82.31%
	ResNet-32	468k	99.68%	92.36%	86.00%
Udacity	LeNet	54k	0.0304	0.0272	0.0610
	Network-in-Network	965k	0.0603	0.0614	0.0504
	ResNet-32	470k	0.0117	0.0102	0.0959

B. Test Suite Construction

We compare the efficacy of coverage criteria indirectly by comparing the efficacy of test suites that are constructed with the help of each coverage criterion. To simulate a realistic setting, we first construct a large pool of test cases called *master suite* from which to construct multiple subsets for each criterion. With each criterion, test cases are picked repeatedly in a random order, included in the test suite if the new test case contributes to increasing the coverage, and discarded if not. This greedy algorithm does not ensure the minimality of the down-selected test suites, but it resembles realistic use case and widely adopted in existing testing research [49]–[51].

For a fair comparison among coverage criteria, we control every other factor besides coverage-based test selection. The controlled variables include the number of test cases in the down-selected suites and the order in which the selection is performed. The order of selection is decided randomly once and for all and shared. The number of test cases is capped because the failure-revealing efficacy of a test suite can depend on the test suite size [52]—if a criterion simply decides to include every test case, the down-selected suite will be the same as the master suite and include the highest number of failure-revealing test cases. This means that we did not aim at creating suites that achieve 100% coverage for a given criterion, because in practice, unless the coverage criterion is known to be very easy to achieve like neuron coverage, it is rarely feasible to achieve 100%. By capping the number of test cases, we instead simulate a more common testing situation where the budget is limited, either for the test execution or for the analysis, and compare the relative efficacy.

As we aim to test the DNNs for generalization, we constructed the master suite by combining multiple datasets that present sufficient challenges. For MNIST, we used EMNIST [53] (an extension of MNIST), and MNIST-C [54], a corrupted variant. For CIFAR, we used CIFAR-10.1 [7] which is independently collected while matching the CIFAR-10 data distribution, and also CIFAR-10-c [15], a corrupted variant. For the Udacity dataset, we used an independently collected dataset shared by Udacity [5] that consists of 55,262 data points. This test data is quite different from the training data, being collected from different circumstances. The distribution shift can be observed both visually and in the output distribution, although the trained models generalize fairly well.

In the case of MNIST and CIFAR-10, combining all the available generalization and corruption datasets yield more than one million data points, but we randomly sampled 90,500 for MNIST and 88,000 for CIFAR-10 in order to keep the computation tractable. The limitation was imposed largely due to the space complexity of white-box criteria, for keeping track of millions of coverage obligations for every test case.

C. Experiment Configurations

Since DNNs lack convenient logic structure, most of the coverage criteria require configuration. As such, studying the impact of all the different configurations of each structural criterion is a challenging task, requiring an extensive empirical validation on its own. We instantiate a small number of representative configurations for each criterion.

Neuron-level Coverage: To allow for differentiating at least the number of test cases that we desire to collect—10,000 test cases, in our experiments—we instantiated NC with higher threshold values—1.5, 3.0, and 4.0. When the threshold value is set to a low value like 0, it becomes too easy to cover a majority of neurons with a small number of test cases. The implication of choosing the threshold value may change for a different DNN architecture, and it is especially susceptible to the choice of activation functions and batch normalization, as those dictate the range of intermediary neuron outputs. As KMNC is granular than NC, it can better differentiate test cases with higher setting of k. However, this merit makes KMNC become prohibitively expensive as the network grows larger in size; state-of-the-art CNNs are often comprised of as many as hundreds of millions of neurons. As an example, if we attempt to measure $k = 100$ MNC on a network with 100 million neurons, the number of coverage obligation becomes ten billion, which amounts to 1.25GB of storage space for strong a single bit vector that represents the coverage of a single test case. For these reasons, we limited our experiments to relatively small k values of 3, 10, and 20. We also limited the length of the coverage vector by measuring both NC and MNC on the last few layers of the larger neural networks—every layer for LeNet, last 7 layers for NiN, and last 11 layers for ResNet—since, those layers in CNNs are known to encode higher-level features [55]. Further investigation is needed to study the impact of the choice of layers.

Surprise Adequacy: SA is configurable in two main ways—selection of layer for the activation trace, and the granularity of coverage. For layer, we chose the last fully-connected layer before the output layer. For granularity, which determines how many test cases can be uniquely identified, we set it proportional to the number of test cases we desire to select for constructing a test suite. In order to ensure that the desired number of test cases are selected even when a test input does not exist for a specific surprise segment, we instantiated DSA1 with $n \times 2$ segments, and another $DSA2$ with $n \times 3$ segments, where n is the testing budget.

Manifold Combination Coverage: For the implementation of the VAE, we revised the open-sourced TensorFlow code [56] written by Dai *et al*. [57]. We used the same encoder and the decoder architecture inspired by InfoGAN [58] for the three datasets, and they are mainly composed of convolution and transposed convolution layers. The number of VAE parameters range from 13 to 38 million. We set the manifold dimension d to 7 for all three datasets. For configuring MCC, we instantiated two versions—$(5, 7)$-MCC and $(6, 7)$-MCC. These numbers were empirically determined based on our budget. The implication of changing these parameters is beyond the scope of this study.

Miscellaneous: For answering RQ3, we used the same optimizer and the learning rate last used for training the original model, and retrained for 20 epochs. We also split the master suite into a selection pool and a test set by 1:2 ratio, so that the retraining accuracy can be assessed with a separate set-aside dataset. The size of test suite T was fixed at 1,000.

VII. Results and Discussion

We present the raw data in Table II. The columns show the coverage criterion used for constructing each test suite, and the rows show the specific MUT. The score is measured for the three evaluation criteria we proposed, and they are presented together by three groups of nine rows. In each row, the highest scores with and without prioritization are separately highlighted with bold-face, and the higher score (lower for MAE) between the two is underlined. Especially, to illustrate how the failure-revealing effectiveness changes as the size of the test suite grows, we visualized the correlation in Table III. The efficacy with and without prioritization for the same setting is presented in the *Default* and *Prioritized* columns, respectively. The x-axis shows the test suite size as it grows by the coverage-guided selection, and the y-axis shows the cumulative number of failure-revealing test cases. The baseline (solid black) is a random selection without any guidance—with a truly random selection, the slope is equal to the percentage of failure-revealing test cases in the master suite. A coverage criterion is deemed effective if its average slope is greater than that of the baseline; the steeper the slope, the better. A line that stops before reaching the end ($x = 10000$) means that the criterion failed to find additional test cases from the master suite to improve coverage. This is generally undesirable, since it is a sign that the specific instance of a criterion is too coarse.

	Model	No Prioritization												With Prioritization										
		Rand	MCC		DSA		NBC	kMNC			NC			None	MCC		DSA		NBC	kMNC			NC		
			1	2	1	2		3	10	20	1.5	3.0	4.5		1	2	1	2		3	10	20	1.5	3.0	4.5
Failure-Revealing	M-L	.110	.071	.071	**.207**	.151	.014	.004	.018	.038	.001	.005	.005	**.410**	.209	.272	.239	.306	.025	.033	.099	.163	.012	.020	.023
	M-N	.115	.075	.077	**.244**	.191	.063	.022	.089	.107	.009	.013	.015	**.310**	.201	.247	.266	.295	.123	.087	.232	.290	.032	.040	.045
	M-R	.110	.077	.076	**.266**	.259	.045	.010	.039	.077	.019	.021	.019	**.409**	.221	.289	.263	.349	.116	.067	.180	.268	.067	.079	.069
	C-L	.114	**.117**	.115	.109	.114	.005	.013	.040	.073	.001	.004	.008	**.245**	.214	.232	.117	.160	.007	.041	.101	.164	.004	.016	.020
	C-N	.011	.114	.112	**.155**	.140	.039	.043	.093	.100	.012	.020	.024	**.372**	.297	.347	.253	.319	.050	.218	.337	.360	.060	.096	.118
	C-R	.112	.113	.116	**.178**	.138	.039	.036	.090	.101	.019	.037	.032	**.433**	.348	.402	.273	.351	.058	.214	.389	.414	.089	.163	.151
	U-L	.160	.345	**.467**	-	-	.204	.198	.199	.166	-	-	-	**.656**	.402	.517	-	-	.248	.363	.523	.599	-	-	-
	U-N	.174	.500	**.650**	-	-	.354	.320	.296	.226	-	-	-	**.783**	.535	.686	-	-	.411	.417	.631	.709	-	-	-
	U-R	.171	.309	**.400**	-	-	.155	.165	.350	.251	.100	.063	.044	**.557**	.311	.410	-	-	.159	.226	.398	.388	.125	.082	.046
Semantic Balance	M-L	.978	.951	**.976**	.921	.946	.848	.870	.865	.878	.796	.831	.895	.835	**.904**	.869	.766	.823	.790	.814	.828	.836	.713	.796	.836
	M-N	.980	.953	**.979**	.925	.953	.874	.910	.918	.953	.902	.921	.894	.865	**.911**	.875	.825	.871	.886	.864	.886	.881	.815	.856	.858
	M-R	.976	.976	.975	**.977**	.976	.967	.969	.972	.977	.960	.969	.951	.979	.977	**.982**	.972	.975	.980	.971	.977	.974	.961	.971	.972
	C-L	.990	.990	.989	.985	**.991**	.045	.972	.985	.989	.907	.946	.955	.806	.848	.819	.888	**.892**	.800	.864	.873	.886	.857	.820	.845
	C-N	.986	.989	.989	**.991**	.987	.989	.978	.988	.989	.943	.968	.975	.784	.852	.824	**.881**	.821	.855	.843	.822	.809	.835	.835	.847
	C-R	.985	.950	.977	.967	**.981**	.909	.917	.958	.972	.928	.924	.914	.897	.915	**.989**	.931	.926	.898	.885	.893	.894	.894	.898	9.883
	U-L	.637	**.807**	.770	-	-	.686	.684	.655	.645	-	-	-	.643	**.805**	.770	-	-	.694	.701	.675	.674	-	-	-
	U-N	.642	**.803**	.764	-	-	.689	.724	.675	.654	-	-	-	**.868**	.835	.789	-	-	.721	.837	.820	.846	-	-	-
	U-R	.641	**.797**	.766	-	-	.699	.753	.733	.687	.733	.761	.759	.689	**.808**	.769	-	-	.708	.771	.749	.720	.758	.777	.765
Retraining Efficacy	M-L	.990	.988	.988	.990	**.991**	.989	.987	.990	.990	.986	.987	.987	.990	.988	.988	.990	.990	.989	.989	.990	.990	.988	.988	.988
	M-N	.984	.987	.988	**.992**	.991	.990	.989	.989	.989	.989	.989	.990	.988	.987	.986	.988	.987	.987	.986	.986	.987	.989	**.989**	.987
	M-R	.994	.989	.991	.995	**.995**	.994	.992	.994	.994	.993	.993	.993	.996	.996	.996	.997	.997	**.998**	.996	.997	.996	.997	.997	.997
	C-L	.937	.937	.937	.937	.937	.937	.937	.937	.937	.937	.937	.937	.937	.937	.937	.937	.937	.937	.937	.937	.937	.937	.937	.937
	C-N	.956	.957	.956	.956	.956	.955	.957	.956	.957	.956	.956	.956	.957	.957	.957	.957	.957	.957	.957	.957	.957	.957	.957	.957
	C-R	.961	.960	.960	**.962**	.961	.959	.961	.960	.961	.961	.960	.960	.965	.966	.966	.967	.967	.969	.969	.969	.969	.970	.970	**.970**
	U-L	.088	.088	.083	-	-	.740	.760	.074	**.068**	-	-	-	.094	.085	.091	-	-	.083	.084	.075	**.075**	-	-	-
	U-N	.039	.039	.039	-	-	.039	.039	.039	.039	-	-	-	.041	.041	.041	-	-	.041	.041	.041	.041	-	-	-
	U-R	.057	.052	.049	-	-	.049	.048	.047	.046	.046	.046	**.045**	.050	.048	.048	-	-	.048	.046	.042	**.041**	.043	.043	.042

TABLE II: Three measures of effectiveness of each criterion (column), for each model under test (row). The acronyms used in the *Model* column stand for M: MNIST, C: CIFAR-10, U: Udacity, and L: LeNet, N: Network-in-Network, R: ResNet.

A. RQ1: Failure-Revealing Effectiveness

Table III shows the correlations of test suites with different coverage scores to the failure-revealing effectiveness of the test suites. Irrespective of test prioritization, most of the lines are above the baseline, indicating that coverage is positively correlated with failure-revealing effectiveness. The relative effectiveness of coverage criteria was not stable across different configurations. We will visit them case by case.

In the case of MNIST and CIFAR-10, without prioritization, the effectiveness of criteria were roughly in the order of DSA1, DSA2, and the rest. No white-box criterion purely based on the structure was meaningfully more effective than random baseline. In the case of Udacity, without prioritization, the effectiveness of criteria were roughly in the order of MC2, MC1, NBC, KMNC, and others. Here, the most effective criterion was manifold coverage, with lower granularity MC1 being more efficient when the suite size is small, but plateauing quickly just like with DSA, and DSA2 eventually becomes more effective. One of the reasons why Udacity results look very different from those of MNIST and CIFAR-10 is due to the unique setting of Udacity test dataset, where the training dataset and the master suite were set up differently to simulate the scenario of a model being evaluated for its generalization to a different environment. In these cases, coverage criteria that are designed to incentivize novelty beyond training dataset—such as MC1, MC2, NBC, N4.5—performed well. The precedence among coverage criteria stays relatively consistent across different models, except for $k = 3$ KMNC in *U-R–Default*. Although the clear reason is not known, we conjecture that the batch normalization [59] used uniquely in ResNet is normalizing the distribution of the internal neuron outputs, and that is positively affecting KMNC. For other criteria such as NC, the efficiency was very high in the beginning, as can be seen from the steepness of the slopes, but was not able to differentiate after a certain point. It is likely because NC is too easily achieved with a small number of test cases, that the small number of remaining obligations are satisfied only with corner-cases, which translated to initially high failure-revealing effectiveness.

Most notably, a stark difference is observed between the results with prioritization and the results without. In general, the presence of prioritization was a much stronger contributor to the effectiveness of a test suite than coverage criterion, the effectiveness changing as much as 243% with prioritization (*C-R*). With prioritization, combination with *any* coverage criterion negatively affected the failure-revealing effectiveness. In summary, **the failure-revealing effectiveness of MCC is comparable to those of white-box criteria**. Also, when it comes to the failure-revealing effectiveness alone, **test prioritization was much more effective than coverage-based test selection** for test suites of the same size.

B. RQ2: Semantic Balance

A coverage criterion can be deemed to contribute to semantic balance when the balance score achieved by a test suite created with the criterion is higher than when the test

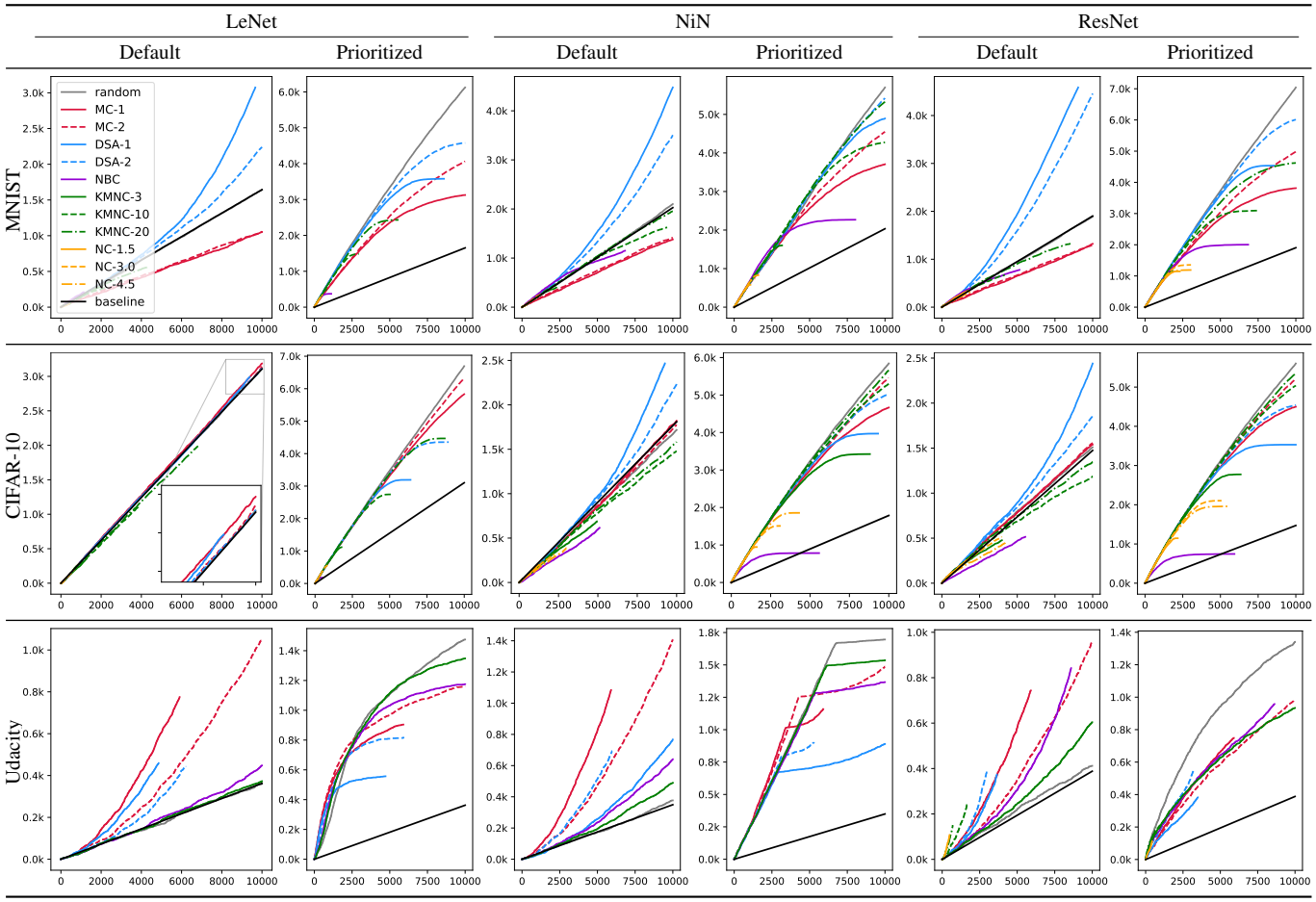

TABLE III: Failure-revealing effectiveness of test suites constructed by each coverage criterion. The x-axis shows the size of the test suite, and the y-axis shows the number of failure-finding test cases in each test suite.

suite is constructed randomly. The *semantic balance* for each configuration is presented in the second rows of each model in Table II. The balance score of random test suites are presented in *Rand* column, and the scores of suites that are constructed only with prioritization is presented in *None* column. The scores of random suites for MNIST and CIFAR-10 are close to 1.0 since the master suite contains exactly the same number of test cases per each class. The scores of random suites for Udacity, on the other hand, are much lower than 1.0 at around 0.64, showing that the master suite is skewed when compared to the training dataset. For every case, the balance score decreased with prioritization, as shown in *None* column. The white-box criteria were not particularly helpful in creating a balanced test suite, whether with or without prioritization. Some criteria such as NBC and NC negatively impacted the balance in many cases. MCC, on the other hand, created test suites of higher balance scores than *Random*; especially in Udacity, the scores were around 0.8, where as the random suite scored around 0.64. MCC was also effective in preserving the semantic balance of the test suites under prioritization, even under a shift in distribution. Other white-box criteria, on the other hand, were shown to be less effective. In summary **MCC is more effective in creating semantically balanced test suites than white-box criteria regardless of prioritization**.

C. RQ3: Retraining Efficacy

The third rows for each model in Table II present the performance of the model measured as classification accuracy or MAE after retraining with the test suites constructed by each criterion. The difference compared to baseline may seem trivial for some cases, but even a small difference may translate to a larger difference in actual operational environment, as demonstrated in experiments by Rechet *et al.* [7], [60]. Without prioritization, the retrained models were more effective when tests were selected with coverage criterion. For MNIST and CIFAR-10, DSA showed the highest retraining efficacy whereas MCC showed the lowest. Unlike in failure-revealing effectiveness where prioritization had a huge impact, the retraining efficacy were not consistently better with prioritization. We conjecture that this phenomenon is caused by the prevalence of failures that are caused by the same *faults*, but the exact reason is unknown. The relative efficacy among criteria could not be determined neither. In summary, **coverage-guided test selection, on average, showed marginally higher retraining efficacy compared to random selection, but MCC was worse than white-box criteria on average. No criterion was shown to perform consistently better than others**.

D. Discussion

Although our RQs were designed to compare our proposed approach against others, our experiments revealed that there are more fundamental issues with coverage criteria—their lack of consistent effectiveness. We observed that many of the test suites constructed with most of the white-box criteria except DSA are not consistently more effective than random suites of the same size. We also observed that test prioritization for DNNs does much of what we hope for from a coverage criterion—help in constructing effective test suites. This, however, does not completely discredit the utility of coverage, as test prioritization assumes that one has access to more tests than needed, and it cannot not quantify the quality of a test suite. It was also shown that prioritized test suites were not always more effective in retraining efficacy, contrary to the results for failure-revealing effectiveness, indicating that a superior failure-revealing effectiveness do not necessarily translate to a better fault detection.

To discuss more about MCC, what is the implication of the results on its practical utility? MCC showed advantage over white-box criteria on semantic balance. When combined with prioritization, its failure-revealing effectiveness was among the highest. Given these results and the unique benefits of black-box testing, MCC combined with prioritization can be one of the best low-cost choice for achieving high failure-revealing effectiveness and semantic balance. It is also possible that a combination of these orthogonal approaches—black-box testing, white-box testing, and test prioritization—may help one construct test suites that are effective in every regards.

E. Threats to Validity

External: While we tried to ensure the variety of subjects, they were limited to specific choices of task and DNN architecture. Our findings may not generalize beyond tested configuration, and especially should not be expected to generalize beyond image classification and regression models. However, we believe that the choice of DNN architecture and the techniques employed within those networks is quite representative for image classification and regression tasks.

Internal: As can be seen from the result that test prioritization changes the efficacy of criteria, the ordering of test case selection matters, and our comparison is not free from the effect of ordering. Also, the upper bound on test suite size was arbitrarily chosen. The efficacy of the coverage criteria was shown to be highly sensitive to many factors such as the type of task, the choice of test dataset, the architecture of the MUT, and configuration of coverage criteria. For the scope of the present work, we did not attempt to reveal the full nuances; a more elaborate study is required to understand the influence of all the factors on the efficacy of the coverage criteria.

VIII. Related Work

Coverage Criteria for DNNs: With the increasing use of deep learning in the real-world, the need for testing learning-enabled systems and measuring the adequacy arose rapidly [8] leading to the recent development of a host of new coverage criteria [21]–[30]. The efficacy of these criteria were evaluated in their own contexts—such as adversarial testing or synthetic test input generation—and thus it is not possible to answer which criterion is the most suited for a given application context. Li *et al*. [61] raised questions about using NC for adversarial testing, showing that the high (adversarial) failure-revealing capability conjectured for neuron coverage is not due to the strength of the criterion. Harel-Canada *et al*. [62] further investigated the utility of NC for synthesizing test inputs, and showed that it is *n*egatively correlated with characteristics desired of a synthesized input, such as failure-revealing efficacy and naturalness. Abrecht also studied the effectiveness of NC in test generation and reached a similar conclusion [63]. However, none of the works have compared the effectiveness of the coverage criteria in the context of test suite construction.

Testing for Generalization: Since the discovery of adversarial examples [64], [65], adversarial attacks and defenses have been hot research topics [12], [66]. However, adversarial robustness determination based on deliberately created attacks does not necessarily translate to better performance of the models on real-world data distribution [18], and thus recently generalization has been attracting more attention under different names: common corruption robustness [15], generalization [7], [60], and natural adversarial examples [18]. Some recent works acknowledge the importance of generalization to realistic real-world data, and either attempt to generate new test cases [67], [68] or measure the efficacy of coverage criterion for realistic test inputs [30]. We believe that testing research should pay more attention to testing for the generalization of the model rather than contending with the discovery of adversarial or unrealistic test inputs, as evidence suggests that adversarial examples are pervasive and no practical defense method exists yet [61], [69].

Manifold-based Approaches: the idea of using manifold for assurance activities, including as a means to assess coverage, is proposed [33], and explored in multiple works [68], [70], [71]. Byun *et al*. [68] explored the idea of generating test cases from manifold using VAE, but the manifold was searched randomly without any notion of coverage over it.

IX. Conclusion

We provided a rigorous definition a new black-box coverage criterion for testing ML systems and systematically evaluated its effectiveness by establishing metrics for assessing coverage criteria on generic properties of interest. Empirical comparison with white-box criteria showed that the new criterion is effective for creating a semantically-balanced test suite with similar fault-revealing ability. The experiments also revealed the weaknesses of ML coverage criteria in general, and the need to further investigate the impact of various factors that influence their effectiveness, an area for future work.

Acknowledgment

This work was supported by AFRL and DARPA under contract FA8750-18-C-0099.

REFERENCES

[1] A. Rajan, "Coverage metrics to measure adequacy of black-box test suites," in *21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06)*, 2006, pp. 335–338.

[2] M. W. Whalen, A. Rajan, M. P. Heimdahl, and S. P. Miller, "Coverage metrics for requirements-based testing," in *Proceedings of the 2006 International Symposium on Software Testing and Analysis*, ser. ISSTA'06. New York, NY, USA: ACM, 2006, pp. 25—-36.

[3] Y. LeCun, "The MNIST database of handwritten digits," 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[4] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[5] Udacity, "Become a self-driving car engineer," https://github.com/udacity/self-driving-car, 2015.

[6] T. Byun, V. Sharma, A. Vijayakumar, S. Rayadurgam, and D. Cofer, "Input prioritization for testing neural networks," in *2019 IEEE Intl. Conference On Artificial Intelligence Testing (AITest)*, April 2019, pp. 63–70.

[7] B. Recht, R. Roelofs, L. Schmidt, and U. C. Berkeley, "Do CIFAR-10 Classifiers Generalize to CIFAR-10 ?" pp. 1–25, 2018.

[8] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *CoRR*, vol. abs/1906.10742, 2019.

[9] N. Humbatova, G. Jahangirova, G. Bavota, V. Riccio, A. Stocco, and P. Tonella, "Taxonomy of real faults in deep learning systems," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1110–1121. [Online]. Available: https://doi.org/10.1145/3377811.3380395

[10] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software engineering for machine learning: A case study," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2019, pp. 291–300.

[11] N. B. Ruparelia, "Software development lifecycle models," *SIGSOFT Softw. Eng. Notes*, vol. 35, no. 3, p. 8–13, May 2010. [Online]. Available: https://doi.org/10.1145/1764810.1764814

[12] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," 2016.

[13] A. Shahrokni and R. Feldt, "A systematic review of software robustness," *Information and Software Technology*, vol. 55, no. 1, pp. 1–17, 2013.

[14] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Towards proving the adversarial robustness of deep neural networks," *Electronic Proceedings in Theoretical Computer Science*, vol. 257, p. 19–26, Sep 2017. [Online]. Available: http://dx.doi.org/10.4204/EPTCS.257.3

[15] D. Hendrycks and T. Dietterich, "Benchmarking neural network robustness to common corruptions and perturbations," in *7th International Conference on Learning Representations, ICLR 2019*, 2019, pp. 1–16.

[16] K. Kawaguchi, L. P. Kaelbling, and Y. Bengio, "Generalization in deep learning," 2020.

[17] M. Alzantot, Y. Sharma, A. Elgohary, B.-J. Ho, M. Srivastava, and K.-W. Chang, "Generating Natural Adversarial Examples," no. 2016, pp. 1–15, 2018. [Online]. Available: http://arxiv.org/abs/1804.07998

[18] D. Hendrycks, K. Zhao, S. Basart, J. Steinhardt, and D. Song, "Natural Adversarial Examples," 2019. [Online]. Available: http://arxiv.org/abs/1907.07174

[19] B. Beizer, *Software testing techniques*. Dreamtech Press, 2003.

[20] K. Pei, Y. Cao, J. Yang, and S. Jana, "DeepXplore: Automated Whitebox Testing of Deep Learning Systems," in *SOSP '17*. New York, New York, USA: ACM Press, 2017, pp. 1–18.

[21] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu, J. Zhao, and Y. Wang, "Deepgauge: Multi-granularity testing criteria for deep learning systems," in *Proceedings of the 33rd ACM/IEEE Intl. Conf. on Automated Software Engineering*, ser. ASE 2018. New York, NY, USA: ACM, 2018, pp. 120–131.

[22] J. Kim, R. Feldt, and S. Yoo, "Guiding deep learning system testing using surprise adequacy," in *Proceedings of the 41st International Conference on Software Engineering*, ser. ICSE '19. IEEE Press, 2019, pp. 1039–1049.

[23] D. Wang, Y. Chen, and Z. Chen, "DeepPath : Path-driven Testing Criteria for Deep Neural Networks," *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)*, pp. 119–120, 2019.

[24] L. Ma, F. Juefei-Xu, M. Xue, B. Li, L. Li, Y. Liu, and J. Zhao, "Deepct: Tomographic combinatorial testing for deep learning systems," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Feb 2019, pp. 614–618.

[25] Y. Sun, M. Wu, W. Ruan, X. Huang, M. Kwiatkowska, and D. Kroening, "Concolic testing for deep neural networks," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE 2018. New York, NY, USA: ACM, 2018, pp. 109–119.

[26] Y. Sun, X. Huang, and D. Kroening, "Testing deep neural networks," *CoRR*, vol. abs/1803.04792, 2018.

[27] X. Du, X. Xie, Y. Li, L. Ma, J. Zhao, and Y. Liu, "Deepcruiser: Automated guided testing for stateful deep learning systems," 2018.

[28] J. Sekhon and C. Fleming, "Towards improved testing for deep learning," in *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, 2019, pp. 85–88.

[29] L. Ma, F. Zhang, M. Xue, B. Li, Y. Liu, J. Zhao, and Y. Wang, "Combinatorial Testing for Deep Learning Systems," pp. 1–14, jun 2018.

[30] S. Gerasimou, H. F. Eniser, A. Sen, and A. Cakan, "Importance-driven deep learning system testing," 2020.

[31] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2020.

[32] M. P. E. Heimdahl, D. George, and R. Weber, "Specification test coverage adequacy criteria = specification test generation inadequacy criteria," in *Eighth IEEE International Symposium on High Assurance Systems Engineering, 2004. Proceedings.*, 2004, pp. 178–186.

[33] T. Byun and S. Rayadurgam, "Manifold for machine learning assurance," 2020.

[34] D. R. Kuhn, I. D. Mendoza, R. N. Kacker, and Y. Lei, "Combinatorial coverage measurement concepts and applications," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, 2013, pp. 352–361.

[35] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, Aug 2013.

[36] C. Doersch, "Tutorial on variational autoencoders," *ArXiv*, 2016.

[37] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[38] A. Groce, M. A. Alipour, and R. Gopinath, "Coverage and its discontents," in *Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software*, ser. Onward! 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 255–268. [Online]. Available: https://doi.org/10.1145/2661136.2661157

[39] I. Ahmed, R. Gopinath, C. Brindescu, A. Groce, and C. Jensen, "Can testedness be effectively measured?" in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 547–558. [Online]. Available: https://doi.org/10.1145/2950290.2950324

[40] M. Papadakis, D. Shin, S. Yoo, and D. Bae, "Are mutation scores correlated with real fault detection? a large scale empirical study on the relationship between mutants and real faults," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, 2018, pp. 537–548.

[41] Y. Feng, Q. Shi, X. Gao, J. Wan, C. Fang, and Z. Chen, "Deepgini: Prioritizing massive tests to enhance the robustness of deep neural networks," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 177–188. [Online]. Available: https://doi.org/10.1145/3395363.3397357

[42] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16. JMLR.org, 2016, pp. 1050–1059.

[43] A. Kendall and Y. Gal, "What uncertainties do we need in bayesian deep learning for computer vision?" *CoRR*, vol. abs/1703.04977, 2017.

[44] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, 2016, pp. 265–283.

[45] W. Li, "cifar-10-cnn: Play deep learning with cifar datasets," https://github.com/BIGBALLON/cifar-10-cnn, 2017.

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

[47] M. Lin, Q. Chen, and S. Yan, "Network in network," 2013.

[48] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[49] M. Staats, G. Gay, M. Whalen, and M. Heimdahl, "On the danger of coverage directed test case generation," in *Fundamental Approaches to Software Engineering*, J. de Lara and A. Zisman, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 409–424.

[50] G. Gay, M. Staats, M. Whalen, and M. P. E. Heimdahl, "The risks of coverage-directed test case generation," *IEEE Transactions on Software Engineering*, vol. 41, no. 8, pp. 803–819, 2015.

[51] T. Byun, V. Sharma, S. Rayadurgam, S. McCamant, and M. P. E. Heimdahl, "Toward rigorous object-code coverage criteria," in *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*, 2017, pp. 328–338.

[52] L. Inozemtseva and R. Holmes, "Coverage is Not Strongly Correlated with Test Suite Effectiveness," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, pp. 435–445. [Online]. Available: https://doi.org/10.1145/2568225.2568271

[53] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "EMNIST: an extension of MNIST to handwritten letters," *CoRR*, vol. abs/1702.05373, 2017.

[54] N. Mu and J. Gilmer, "Mnist-c: A robustness benchmark for computer vision," 2019.

[55] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds. Cham: Springer International Publishing, 2014, pp. 818–833.

[56] B. Dai and D. P. Wipf. (2019) Twostagevae. [Online]. Available: https://github.com/daib13/TwoStageVAE/

[57] ——, "Diagnosing and enhancing VAE models," *CoRR*, vol. abs/1903.05789, 2019.

[58] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "Infogan: Interpretable representation learning by information maximizing generative adversarial nets," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 2172–2180.

[68] T. Byun, A. Vijayakumar, S. Rayadurgam, and D. Cofer, "Manifold-based test generation for image classifiers," *2020 IEEE International Conference On Artificial Intelligence Testing (AITest)*, Aug 2020. [Online]. Available: http://dx.doi.org/10.1109/aitest49225.2020.00010

[59] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015.

[60] B. Recht, R. Roelofs, L. Schmidt, and V. Shankar, "Do ImageNet Classifiers Generalize to ImageNet?" pp. 1–76, 2019. [Online]. Available: http://arxiv.org/abs/1902.10811

[61] Z. Li, X. Ma, C. Xu, and C. Cao, "Structural coverage criteria for neural networks could be misleading," in *ICSE–NIER '19*. Piscataway, NJ, USA: IEEE Press, 2019, pp. 89–92.

[62] F. Harel-Canada, L. Wang, M. A. Gulzar, Q. Gu, and M. Kim, "Is Neuron Coverage a Meaningful Measure for Testing Deep Neural Networks?" 2020.

[63] S. Abrecht, M. Akila, S. Gannamaneni, K. Groh, C. Heinzemann, S. Houben, and M. Woehrle, "Revisiting neuron coverage and its application to test generation," 09 2020.

[64] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[65] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 427–436.

[66] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 9, pp. 2805–2824, 2019.

[67] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE 2018. New York, NY, USA: ACM, 2018, pp. 132–142.

[69] A. Fawzi, H. Fawzi, and O. Fawzi, "Adversarial vulnerability for any classifier," in *Advances in neural information processing systems*, 2018, pp. 1178–1187.

[70] S. Han, J. Kim, G. Kim, J. Cho, J. Kim, and S. Yoo, "Preliminary evaluation of path-aware crossover operators for search-based test data generation for autonomous driving," in *2021 IEEE/ACM 14th International Workshop on Search-Based Software Testing (SBST)*, 2021, pp. 44–47.

[71] S. M. Katz, A. L. Corso, C. A. Strong, and M. J. Kochenderfer, "Verification of image-based neural network controllers using generative models," 2021.