An overview of structural coverage metrics for testing neural networks

Muhammad Usman\textsuperscript{1} · Youcheng Sun\textsuperscript{2} · Divya Gopinath\textsuperscript{3} · Rishi Dange\textsuperscript{4} · Luca Manolache\textsuperscript{5} · Corina S. Păsăreanu\textsuperscript{6}

Accepted: 18 October 2022 / Published online: 2 November 2022 © The Author(s), under exclusive licence to Springer-Verlag GmbH Germany, part of Springer Nature 2022

Abstract

Deep neural network (DNN) models, including those used in safety-critical domains, need to be thoroughly tested to ensure that they can reliably perform well in different scenarios. In this article, we provide an overview of structural coverage metrics for testing DNN models, including neuron coverage, \( k \)-multisection neuron coverage, top-\( k \) neuron coverage, neuron boundary coverage, strong neuron activation coverage and modified condition/decision coverage. We evaluate the metrics on realistic DNN models used for perception tasks (LeNet-1, LeNet-4, LeNet-5, ResNet20) including networks used in autonomy (TaxiNet). We also provide a tool, DNNCov, which can measure the testing coverage for all these metrics. DNNCov outputs an informative coverage report to enable researchers and practitioners to assess the adequacy of DNN testing, to compare different coverage measures, and to more conveniently inspect the model’s internals during testing.

Keywords Coverage · Neural networks · Testing

1 Introduction

Today’s world has seen a significant rise in deep learning models that are used for solving complex tasks, such as medical diagnosis, image and video recognition, text processing, program understanding, and also perception and control of self-driving vehicles, since such models are increasingly used for safety-critical applications, ensuring that deep learning models perform as expected is of extreme importance. Typically, statistical accuracy on a separate dataset is used as a measure of model performance. However, it is difficult to guarantee that the set covers all the possible behaviors of the model, including corner cases. Formal methods are traditionally used to provide such guarantees; however, they are computationally infeasible for most realistic neural networks. While it is not as rigorous as formal methods, software testing can be used for ensuring trustworthiness of software, where coverage metrics can be used to measure the adequacy of testing. In this article, we review and evaluate recently proposed structural coverage metrics for testing neural networks. In addition, we describe a tool, DNNCov, which incorporates into a common framework these different coverage metrics, to allow for easy experimentation and comparison.

Specifically, we evaluate neuron coverage and its variants, as well as a more complex MC/DC (modified condition/decision coverage) criterion for neural networks. Neuron coverage was first proposed in DeepXplore \cite{1} where it was developed as a DNN counterpart for the statement coverage that is commonly used in testing traditional software. Neuron coverage was later extended to a family of more fine-grained, DNN-specific criteria \cite{2}, such as \( k \)-multisection neuron coverage (KMNC), top-\( k \) neuron coverage (TKNC), top-\( k \) neuron patterns (TKNP), neuron boundary coverage (NBC) and strong neuron activation coverage (SNAC) \cite{2}. MC/DC was originally developed by NASA and has been widely used for testing high integrity software; it was recently adapted for testing DNNs \cite{3}. Subsequently, there has been a boom in coverage-guided DNN testing techniques. ADAPT \cite{4} uses an adaptive learning algorithm to generate new images for testing neural networks with an aim of increasing, e.g., neuron coverage. DeepHunter \cite{5} is a fuzzing framework which

\textsuperscript{1} University of Texas at Austin, Austin, USA
\textsuperscript{2} University of Manchester, Manchester, UK
\textsuperscript{3} KBR Inc. at NASA Ames, Ames, USA
\textsuperscript{4} Princeton University, Princeton, USA
\textsuperscript{5} Palo Alto High School, Palo Alto, USA
\textsuperscript{6} CMU Cylab and KBR Inc. at NASA Ames, Ames, USA
uses metamorphic mutations to generate tests following the metrics in [2] while the concolic testing in [6] targets MC/DC.

Despite the proliferation of the aforementioned approaches, it is still not well understood which technique or criteria can adequately determine whether a neural network model has been well tested with respect to properties such as functional diversity or vulnerability to attacks (e.g., adversarial input perturbations). The problem is challenging due to the opaque nature of the networks. Furthermore, the coverage criteria themselves have little semantic meaning and it is thus difficult to determine whether a given test suite covers adequately the behavior of the neural network. The coverage criterion that is the most effective may vary from one model to another, and from one task to another. Therefore, we conduct a study and present a tool, DNNCov, that given a model and a test suite, evaluates different coverage metrics, thereby enabling a more precise understanding of the model’s performance in relation to the test suite and the considered metrics.

We summarize our contributions as follows:

- We provide an overview of different structural coverage metrics (NC, KMNC, TKNC, NBC, SNAC, and MC/DC) for testing neural networks.
- We evaluate the criteria with respect to functional diversity and defect detection. We consider both adversarial robustness and data poisoning scenarios (the latter has not been considered in previous evaluations). We analyze state-of-the-art models trained for classification and regression tasks. Our evaluation suggests that existing criteria may be inadequate for measuring the covered behavior of neural networks, indicating the need for future testing techniques and tools.
- We describe a tool, DNNCov, that incorporates multiple structural testing criteria, including the complex MC/DC coverage proposed in [3] (not considered in previous evaluations) to facilitate comparison between different testing metrics. DNNCov computes multiple coverage criteria simultaneously giving a 2.5× time improvement over the baseline sequential implementation.
- DNNCov has a visualization component that displays the neural network architecture along with the achieved coverage (neurons covered or not). DNNCov also displays quantitative information, measuring the number of tests that achieve the coverage. We believe that this nuanced view of the coverage can enable developers to better understand and debug the behavior of the neural networks.

1.1 Applications of DNNCov

DNNCov can be used to compare different coverage metrics on the same model and reason about the effectiveness of a metric in determining test set adequacy. Moreover, one can compare the coverage metrics over different models (for the same task) to assess how different architectures impact testing adequacy. One can also compare different test sets with varying sizes to determine how well they test a given model. If a smaller test suite achieves same coverage as that achieved by the full test suite, the test set can also be reduced. Furthermore, DNNCov can enable coverage-driven test generation for different criteria and can be used for optimizing neural networks, for instance, by pruning the neurons that are never covered (or have low coverage).

1.2 Comparison with related studies

There are many approaches and metrics that have been proposed recently, aiming to measure the testing adequacy of DNN models [1–3]. There are also frameworks proposed to facilitate such measurements, such as [7]. In contrast to previous studies, we provide the following contributions. We evaluate different structural metrics with respect to both functional diversity and defect detection. The latter is evaluated not only with respect to adversarial robustness (which is overwhelmingly the only one studied in the related literature), but also with respect to poisoning attacks, which provide additional interesting insights on different coverage metrics. We consider models trained for both classifications and regression tasks. Furthermore, we provide a tool that incorporates these criteria, including the complex MC/DC criterion not considered in previous studies. DNNCov provides finer-grained results, as it computes not only qualitative results (as was done also in previous work) but also quantitative information about how many inputs satisfy the criteria.

2 Structural coverage criteria for neural networks

2.1 Neural networks

Neural networks (NNs) are machine learning algorithms that can be trained to perform different tasks such as classification and regression. NNs consist of multiple layers, starting from the input layer, followed by one or more hidden layers (such as convolutional, dense, activation and pooling), and a final decision layer. Each layer consists of a number of computational units, called neurons. Each neuron applies an activation function on a weighted sum of its inputs (coming from the previous layer). $$N(X) = \sigma(\sum_i w_i \cdot N_i(X) + b)$$ where \(N_i\) denotes the value of the \(i\)th neuron in the previous layer of the network and the coefficients \(w_i\) and the constant \(b\) are referred to as weights and bias, respectively; \(\sigma\) represents the activation function. For instance, the ReLU (rectified linear unit) activation function returns its input as is if it is positive, and returns 0 otherwise, i.e., \(\sigma(X) = \max(0, X)\).
The final decision layer (logits) typically uses a specialized function (e.g., max or `softmax`) to determine the decision or the output of the network.

### 2.2 DNN structural coverage metrics

In this study, we evaluate neuron coverage [1] and its extensions [2] and the MC/DC variants for DNNs [3]. We describe each of these criteria by formulating their test conditions. Given a test suite and a DNN model, the coverage for each criterion is thus computed as the portion of test conditions that are satisfied. In the following, we use $a_{l,i}$ to denote a neuron’s activation value, where $l$ is the layer index and $i$ is the neuron index at that layer; $L$ is the total number of layers in the DNN.

**Neuron coverage (NC)** NC can be seen as a statement coverage variant for DNNs. A neuron $n_{l,i}$ is said to be covered, if its neuron activation value ($a_{l,i}$) is larger than 0 (or some specified threshold) for at least one test input. Thus, the set of test conditions to be met for NC can be formulated as follows.

$$\{a_{l,i} > 0 | 1 < l < L\}$$

**Neuron boundary coverage (NBC)** NBC extends NC by considering the neuron activations at the maximum and minimum boundary cases. Assuming $\text{high}_{l,i}$ and $\text{low}_{l,i}$ are, respectively, the estimated upper and lower bounds on the neuron activation value $a_{l,i}$, we can formulate the set of test conditions for NBC as follows.

$$\{a_{l,i} > \text{high}_{l,i}, a_{l,i} < \text{low}_{l,i} | 1 < l < L\}$$

The estimation of the bounds is typically done via profiling with the training dataset. Intuitively, for new test inputs, the output of the neurons may fall outside the interval $[\text{low}_{l,i}, \text{high}_{l,i}]$ prescribed by the training set, indicating testing of new network behavior.

**Strong neuron activation coverage (SNAC)** SNAC focuses on test conditions on corner cases with respect to the upper boundary value.

$$\{a_{l,i} > \text{high}_{l,i} | 1 < l < L\}$$

**K-multisection neuron coverage (KMNC)** KMNC divides a neuron’s activation range between $\text{high}_{l,i}$ and $\text{low}_{l,i}$ into $K$ equivalent sections, each denoted by $\text{range}_{l,i,k}$, and test conditions in KMNC are defined as the coverage of these activation sections.

$$\{a_{l,i} \in \text{range}_{l,i,k} | 1 < l < L, 1 \leq k \leq K\}$$

**Top-K neuron coverage (TKNC)** Given a test input $x$, a neuron is TKNC covered if its neuron activation value is one of the most active $K$ neurons at its layer, denoted by $a_{l,i} \in \text{top}_{K}(l, x)$. Here $\text{top}_{K}(l, x)$ denotes the neurons that have the largest $K$ activation values at layer $l$ on input $x$. The rationale for this criterion is that top active neurons (at different layers) may be good indicators for major functionality in the network. The test conditions are as follows.

$$\{a_{l,i} \in \text{top}_{K}(l, x) | 1 < l < L\}$$

**Modified condition/decision coverage (MC/DC)** Different from the coverage criteria above, MC/DC takes into account the relation between neuron activations at two adjacent layers, such that its test conditions require that any neuron activation at layer $l + 1$ (decision) must be independently impacted by each neuron at layer $l$ (condition).

$$\{\forall i, j, h, \text{change}(a_{l,i}) \land \text{change}(a_{l+1,j}) \land \neg \text{change}(a_{l,h}) | 1 < l < L - 1\}$$

A Sign (S) change function and a Value (V) change function are defined in [3] for depicting how a neuron activation changes when the input changes from a test to another. As a result, there is a family of four variants of MC/DC for DNNs, including SS coverage (SS), SV coverage (SV), VS coverage (VS) and VV coverage (VV). To ease the use of MC/DC in large DNN models, in the later evaluation, we generalize its test conditions from single neurons to sets of neurons (feature maps) between two adjacent layers.

In this study, we focus on structural coverage criteria for DNNs, due to their popularity, wide use, and similarity to established metrics for general-purpose software. These criteria are thus more likely to be integrated in certification procedures for safety-critical systems that contain neural networks. There are also other, non-structural proposals for measuring the adequacy of testing DNNs, e.g., the safety coverage [8] and the surprise adequacy [9].

### 3 The DNNCov tool

DNNCov (Fig. 1) provides an integrated framework for testing neural network models. The framework takes an input test set and optionally a training set and computes the different coverage results achieved by the set. The user also needs to specify a configuration (# of training/test inputs to be used and criteria to calculate). DNNCov profiles the neural network model to calculate threshold values based on the training set. These threshold values are used as input to the coverage calculation module. The coverage calculation module computes coverage for the different metrics. DNNCov
then produces a summarized coverage report and a visualization of the results.

### 3.1 Implementation, profiling and coverage calculation

To implement DNNCov, we leveraged the DeepHunter [5] codebase, which already provided support for the neuron-based coverage metrics. We extended it with the four MC/DC variants, i.e., sign–sign (SS), sign–value (SV), value–sign (VS) and value–value (VV). We profile the neural network model to estimate the threshold values needed in some coverage criteria. We wrote new functions to measure coverage for a given test suite, replacing the original test generation process. We also implemented simultaneous coverage computations to improve efficiency (achieving $2.5 \times$ runtime improvement). DNNCov outputs an informative coverage report to the user as illustrated in Fig. 2.

### 3.2 Reporting quantitative information

DNNCov not only computes which coverage obligations are fulfilled, but also records the number of tests that achieve the coverage. This quantitative information gives us an idea of which parts of the network have been exercised lesser than others, thus enabling us to direct testing/debugging efforts in those parts. This is specifically important for neural networks since unlike traditional programs, every test exercises almost every neuron of the network and the changes in behavior is due to the different output values of the neurons. Therefore, executing a neuron only once or just few times may not suffice to exercise different behaviors or expose vulnerabilities. Dif-

---

**Fig. 1** DNNCov tool

**Fig. 2** Tool progress on MNIST (LeNet-4)
different test suites may have similar total coverage values but different coverage distributions. This cannot be highlighted by existing coverage metrics. We calculate minimum, maximum, average, standard deviation and variance of the number of inputs that cover each of the coverage obligations for each coverage criterion.

3.3 Visualization

The coverage results computed by DNNCov are saved in a file and are viewed by the visualization component of the tool (which is built based on Tensorflow Playground). This component has a web-based interface that shows the architecture of the model and quantitative coverage information. The user can navigate through the model, and visualize specific information related to the coverage. Neurons are given a color from white (least covered) to red (most covered) based on how many times they are covered (see Fig. 3).

We can see that neurons in layer 5 are covered fewer times as compared to neurons in the layer 1. Moreover, we can observe that neuron 4 in layer 2 is never covered. Either this neuron is not required and it can be pruned from the neural network or the test set has not properly tested this neuron.

Neuron pairs covered according to MC/DC criteria are shown as connected with each other using connection lines. The displayed information can be used to identify poorly tested areas in the neural network.

3.4 Tool configuration

Users can load their own trained neural network model using the --model option. Users can select the coverage criteria by using the --criteria option. There are three possible settings; all calculates all six coverage criteria, no-mcdc excludes MC/DC and mcdc only calculates MC/DC.

The reason MC/DC is separate is that it works on input pairs, whereas the rest of the metrics are computed for each input. Therefore, calculations of MC/DC and the other NC variants are implemented with two separate modules in DNNCov. All output results (including the intermediate files used for visualization) are stored in an output folder. Users can specify the directory of the output folder using the --outputs option. The datasets that are the subject of our study (see the next section) can be automatically loaded in DNNCov using the following:

--mnist-dataset for MNIST,
--cifar10-dataset for CIFAR-10 and
--tinytaxinet-dataset for TaxiNet.

Users’ own datasets can be used by specifying the --input-tests and --input-train folders. The tool also supports calculation over a subset of training and test dataset. Users can specify the number of training/test inputs to be used by using the --train and --test options. For example,

python dnncov.py --criteria all
--model lenet1.h5 --outputs outs1
--mnist-dataset --train 60000
--test 10000

calculates coverage for all six criteria for the lenet1.h5 model using the default training and test sets for MNIST.

4 Evaluation

We structure our evaluation along the following thrusts.

- Comparison of coverage metrics on standard test sets.
- Evaluation of coverage information with respect to sensitivity to functional diversity.
Table 1  Details of the datasets and neural network models

| Model      | Benchmark      | #Dataset (training,test) | Test accuracy | Model architecture                                      |
|------------|----------------|--------------------------|---------------|--------------------------------------------------------|
| LeNet-1    | MNIST          | (60k,10k)                | 90.60%        | 2 conv/2 maxpool                                       |
| LeNet-4    | MNIST          | (60k,10k)                | 89.90%        | 2 conv/2 maxpool/1 dense                               |
| LeNet-5    | MNIST          | (60k,10k)                | 92.80%        | 2 conv/2 maxpool/2 dense                               |
| MNIST-Funct.| MNIST          | (60k,10k)                | 96.34%        | 2 conv/4 act/1 maxpool/2 dense/1 flatten               |
| MNIST-Pois.| MNIST          | (60k,10k)                | Test:98.63, Pois.:10.38% | 2 conv/4 act/1 maxpool/2 dense/1 flatten |
| MNIST-Adv. | MNIST          | (60k,10k)                | Test: 97.87%, Adv.: 28.37% | 2 conv/4 act/1 maxpool/2 dense/1 flatten |
| ResNet20   | CIFAR-10       | (50k,10k)                | 68.70%        | 21 conv/19 batchnorm/19 act/9 add/1 globalavgpool     |
| TinyTaxiNet| TaxiNet        | (51462,7386)             | MAE (CTE:1.44, HE:2.75) | 3 dense                                               |

- Evaluation of coverage information with respect to sensitivity to defect detection; we consider both adversarial and poisoned inputs.
- Evaluation of quantitative information.

In turn, this information can be used by developers to select among multiple test sets, favoring the ones for which multiple metrics yield high coverage. Furthermore, developers can use the coverage information to choose between models (trained for the same task), favoring the model that again achieves high coverage for the majority of metrics. 

**Benchmarks** Table 1 shows the details of the datasets and models used in our study. We used a set of image classification models on benchmark datasets. MNIST (Modified National Institute of Standards and Technology database) [10] is a collection of handwritten digits from ‘0’ to ‘9’. It has a training set (MNIST:Train) with 60k inputs and a test set (MNIST:Test) of 10k inputs, which are 28×28 grayscale images. We used the popular LeNet convolutional neural network model trained for image classification on the MNIST dataset [11,12]. CIFAR-10 (Canadian Institute For Advanced Research) [13] is a collection of color images classified to one of 10 classes which include vehicles such as ‘airplane’ and ‘truck’ and animals such as bird and cat. The training set (CIFAR-Train) contains 50k 32×32 color images and the test set (CIFAR-Test) contains 10k images. We used the state-of-the-art ResNet20 model for image classification on this dataset [14].

We also applied DNNCov on a regression model from the autonomy domain, namely TaxiNet, which is a perception model for centerline tracking in airport runways [15]. The model takes images of the runway as input and produces two outputs, cross-track (CTE) and heading angle (HE) errors which indicate the lateral and angular distance, respectively, of the nose of the plane from the centerline of the runway. We use a model called the Tiny Taxinet [15] which takes in a down-sampled 8×16 image of the runway. It has a training set and test set with 51,462 and 7386 inputs, respectively.

The tool, neural network models along with the datasets and Appendix (which provides more detailed results) are publicly available at the GitHub repository1. All experiments were run on a machine with an Intel Core i9–9980HK processor and 64GB RAM running Windows 10. All experiments were repeated 5 times, and average results are reported.

4.1 Comparison of coverage metrics

Table 2 summarizes the results obtained for different coverage metrics, in terms of percentage of covered obligations, when testing the models using the default test data from MNIST, CIFAR-10 and Tiny TaxiNet. The respective training sets were used to obtain threshold values for the KMNC, NBC, SNAC and MC/DC metrics. As expected, NC (Threshold: 0.00) appears to be the easiest to achieve. The achieved NC coverage is greater than 67% for all the models, with 100% for the LeNet-1 and even the most complex ResNet20 model. Setting the threshold to 0.75 reduces the computed coverage for all of the models.

KMNC (K: 10) assesses whether the tests cover different ranges of neuron values. We can observe that the value for this metric is greater than 85% for the LeNet and ResNet20 models, whereas it is only 62% for Tiny TaxiNet. This appears reasonable considering that the test sets for LeNet and ResNet20 are the standard for the widely studied MNIST and CIFAR-10, respectively, and are thus expected to test the respective models adequately. On the other hand, the test set for Tiny TaxiNet is generated from simulations, which may not have a good coverage of different network behaviors. A similar trend can be observed for TKNC (K: 10).

The NBC and SNAC metrics determine whether there are any tests that exercise neurons beyond the boundaries observed for the training sets. The results indicate that the test sets for the MNIST, CIFAR and Tiny TaxiNet benchmarks are very close in their distribution to the respective training sets, which results in low coverage values for these two metrics.

1 https://github.com/DNNCov/DNNCov
The $MC/DC$ coverage appears to be more difficult to achieve than the other structural metrics. This is expected considering that it involves satisfaction of more constraints. The value–value variant has generally the highest coverage values and is easier to achieve than the sign–sign counterparts. The test set for MNIST has better $MC/DC$ ($VV$) coverage values for LeNet-1 and LeNet-4, whereas the LeNet-5 model with a more complex architecture has the least $MC/DC$ ($VV$) coverage values.

Figure 4 shows in more detail how the metrics compare for the same test set on the same model (LeNet-5). The graphs show how the coverage values change as the number of tests increase. As can be seen the $KMNC$, $NBC$ and $SNAC$ metrics display a more gradual increase in comparison with $NC$ and $TKNC$. This suggests that the $KMNC$, $NBC$ and $SNAC$ metrics may be better for indicating the quality of the test set as compared to the other metrics. For instance, the maximum values for $TKNC$ and $NC$ can be achieved by executing just 10% of the test set. The additional tests, which are redundant with respect to these metrics, are actually useful since they cover different neuron value ranges.

Also note that accuracy on the test sets is typically used to judge the quality of a trained model. Table 1 provides the statistical test set accuracy of the models. For instance, LeNet-5 has 92.8% accuracy while Tiny TaxiNet has Mean Absolute Error for “CTE”: 1.44 and for “HE”: 2.75. A user may consider these models to have good generalization and accuracy on unseen inputs. However, the corresponding coverage measures highlight how much (or how little) this test set accuracy could be trusted. For instance, the Tiny TaxiNet model has low coverage scores indicating that the test set may not have sufficiently tested the behaviors of the model.

4.2 Evaluating coverage metrics with respect to functional diversity

We use each DNN output class as a proxy for the functionality of the model. A dataset is considered to be functionally diverse if it contains inputs belonging to a comprehensive range of output classes. For example, a dataset which consists of inputs belonging to ten output classes has higher functional diversity than a dataset which consists of inputs belonging to only one output class (even if the structural coverage is the same). Intuitively, a functionally diverse test set is better in testing and debugging neural network models, as it covers more behaviors. If a test coverage metric is sensitive to functional diversity, it should obtain higher coverage when the dataset contains inputs belonging to multiple output classes, as opposed to fewer classes.

To evaluate the sensitivity of each metric toward functional diversity, we create six different datasets from the clean MNIST test set (MNIST-Test). We name the datasets as $F_\alpha$ where $\alpha$ is the number of randomly selected output classes whose inputs are included in the respective dataset. For example, $F_1$ represents the dataset which consists of inputs belonging to one randomly selected output class and $F_{10}$ is the dataset which consists of inputs belonging to ten output classes. Similarly, we build $F_2$, $F_3$, $F_5$ and $F_7$. The size of each dataset is fixed to 800.

To compare the different coverage metrics, we normalize the results as follows. For each dataset, we first calculate $\Delta_\alpha$ which is essentially the difference (Eq. 1) between the coverage value of the metric on the baseline dataset ($F_1$) and the respective dataset ($F_\alpha$). $\text{Max}(\Delta_\alpha)$ is the maximum $\Delta_\alpha$ across datasets $F_1$–$F_{10}$. $\text{Min}(\Delta_\alpha)$ is the minimum $\Delta_\alpha$ across datasets $F_1$–$F_{10}$.

| Table 2 Coverage of the neural network models for the standard test sets |
|-----------------|---------------|---------------|---------------|---------------|---------------|
| Coverage metrics (%) | LeNet-1 | LeNet-4 | LeNet-5 | ResNet20 | Tiny TaxiNet |
| $KMNC$ ($K$: 10) | 95.00 | 85.29 | 90.70 | 98.75 | 62.35 |
| $KMNC$ ($K$: 1000) | 60.23 | 54.33 | 59.10 | 71.16 | 43.54 |
| $TKNC$ ($K$: 10) | 88.57 | 81.59 | 82.40 | 65.09 | 52.06 |
| $TKNC$ ($K$: 1000) | 1.00 | 3.27 | 4.93 | 3.90 | 0.59 |
| NBC | 0.87 | 0.66 | 0.58 | 5.55 | 2.01 |
| SNAC | 0.87 | 1.05 | 1.16 | 6.46 | 3.21 |
| $NC$ (Threshold: 0.00) | 100.00 | 90.58 | 96.12 | 100.00 | 67.65 |
| $NC$ (Threshold: 0.20) | 61.90 | 76.09 | 85.66 | 100.00 | 67.65 |
| $NC$ (Threshold: 0.50) | 30.95 | 63.77 | 74.03 | 99.16 | 61.76 |
| $NC$ (Threshold: 0.75) | 23.81 | 59.42 | 67.05 | 13.99 | 52.94 |
| $MC/DC$ (sign–sign) | 5.77 | 44.17 | 10.62 | 24.66 | 27.60 |
| $MC/DC$ (sign–value) | 63.46 | 35.68 | 8.40 | 51.58 | 28.65 |
| $MC/DC$ (value–sign) | 23.08 | 58.20 | 13.71 | 52.20 | 41.67 |
| $MC/DC$ (value–value) | 100.00 | 67.21 | 15.23 | 99.63 | 46.88 |
datasets $F_1$–$F_{10}$. We then calculate normalized difference (NCoverage($F_\alpha$)) in coverage (Eq. 2).

$$\Delta_\alpha = \text{Coverage}_{F_\alpha} - \text{Coverage}_{F_1} \quad (1)$$
$$\text{NCoverage}(F_\alpha) = \frac{\text{Coverage}_{F_\alpha} - \text{Coverage}_{F_1}}{\text{Max}(\Delta_\alpha) - \text{Min}(\Delta_\alpha)} \quad (2)$$

Figure 5b summarizes the results. We consider the coverage obtained on dataset $F_1$ as the baseline. We can see that as the value of $\alpha$ is increased from 1 to 10, the metrics report higher coverage (shown by the high normalized difference value). With the exception of NC (Threshold: 0.00), MC/DC (SS) and MC/DC (SV), all of the remaining 11 metrics/configurations show highest coverage on dataset $F_{10}$. NC (Threshold: 0.00) achieves 100% coverage on all datasets (even when the dataset consists of inputs belonging to only one output class), whereas MC/DC(SS) and MC/DC (SV) obtains 0% coverage on all datasets (even when the dataset has inputs belonging to all output classes). Thus, the results indicate that these three coverage criteria are not adequate for measuring functional diversity.
Figure 5a and Table 3 shows that the accuracy of the neural network model remains between 95% and 98%. Thus, the accuracy metric alone fails to differentiate between these datasets.

### 4.3 Evaluating coverage metrics with respect to defect detection

We evaluate the different metrics by testing the defect detection ability, with respect to adversarial robustness and data poisoning scenarios.

#### 4.3.1 Adversarial scenario

In an adversarial attack [16], a neural network mis-classifies the adversarial input that is obtained by adding small perturbations to the original input. Testing the neural network model with adversarial inputs can help one identify whether the neural network is robust toward adversarial attacks.

For this scenario, we apply perturbations to MNIST images in the training (MNIST-Train) and test sets (MNIST-Test) using the FGSM (Fast Gradient Sign Method) attack [16], with $\epsilon \in \{0.01, 0.05, 0.10, 0.20, 0.30\}$ being used for controlling the perturbation level. This gives us MNIST-Adversarial-Train and MNIST-Adversarial-Test, respectively. For brevity, we report results for $\epsilon = 0.30$ in this article. Results for other values of $\epsilon$ are available in Appendix. Intuitively, if a coverage metric has the capability to assess the defect detection ability of a dataset, one should obtain higher coverage when the dataset contains inputs that can reveal defects in the neural network model. We create dataset $A_0$ by randomly selecting 1000 inputs from the clean MNIST test set (MNIST-Test). We consider the coverage obtained on this dataset as the baseline. From this baseline dataset ($A_0$), we further create six different datasets by replacing $\beta$% of clean inputs with adversarial inputs. We name these datasets as $A_\beta$. For example, $A_1$ represents the dataset in which we replaced 1% (10) of the clean inputs with the same number of randomly selected adversarial inputs and $A_{10}$ is the dataset in which we replaced 10% (100) of the clean inputs with the same number of randomly selected adversarial inputs. Similarly, we have $A_2$, $A_3$, $A_5$ and $A_7$. The size of each dataset is 1000.

For each dataset, we first calculate $\Delta_\beta$ which is essentially the difference (Eq. 3) between the coverage value of the metric on the baseline dataset ($A_0$) and the respective dataset.
Thus, these coverage metrics appear inadequate for measuring the defect detection ability of the dataset. Figure 5a and Table 3 indicate that the accuracy of the model is highest on the clean dataset (A₀). For each dataset, we first calculate \( \Delta_{\beta} \) which is essentially the difference (similar to Eq. 3) between the coverage value of the metric on the baseline dataset (\( P_0 \)) and the respective dataset (\( P_{\beta} \)). \( \max(\Delta_{\beta}) \) is the maximum \( \Delta_{\beta} \) across datasets \( A_0–A_{10} \). \( \min(\Delta_{\beta}) \) is the minimum \( \Delta_{\beta} \) across datasets \( A_0–A_{10} \). We then calculate normalized difference (NCoverage(\( A_{\beta} \))) in coverage (Eq. 4).

\[
\Delta_{\beta} = \text{Coverage}_{A_{\beta}} - \text{Coverage}_{A_0} \tag{3}
\]

\[
\text{NCoverage}(A_{\beta}) = \frac{\text{Coverage}_{A_{\beta}} - \text{Coverage}_{A_0}}{\max(\Delta_{\beta}) - \min(\Delta_{\beta})} \tag{4}
\]

Figure 5c summarizes the results. We can see that as the value of \( \beta \) is increased from 0 to 10, ten metrics (with the exception of \( NC \) (Threshold: 0.00), \( NC \) (Threshold: 0.20), \( NC \) (Threshold: 0.75) and \( MC/DC \) (VV)) report higher coverage (shown by the high normalized difference value). On the other hand, the remaining four metrics (\( NC \) (Threshold: 0.00), \( NC \) (Threshold: 0.20), \( NC \) (Threshold: 0.75) and \( MC/DC \) (VV)) report similar coverage across all datasets. Thus, these coverage metrics appear inadequate for measuring the defect detection ability of the dataset. Figure 5a and Table 3 indicate that the accuracy of the model is highest on the clean dataset (\( A_0 \)) and it gradually decreases as the value of \( \beta \) is increased. This is expected because as the number of adversarial inputs increases in the dataset, there are more inputs on which the neural network fails to predict the correct class.

**4.3.2 Poisoned scenario**

In a data poisoning scenario [17], an attacker has 'poisoned' the training data such that the model has high accuracy on clean data but when it is presented with an input that contains a poisoned 'trigger', the model mis-classifies the respective input to a target output class. Testing the neural network model with poisoned inputs can help one identify whether the neural network contains a backdoor which can be exploited by a malicious actor.

For this scenario, we apply the backdoor attack from [17]. Figure 6 shows some examples of the attack. We applied the backdoor trigger to the first 600 inputs in the MNIST training set (\( MNIST-Train \)); the rest of the 59,400 inputs are not changed. This results in the \( MNIST-Poisoned-Train \) set. For evaluation, we created a poisoned test set (\( MNIST-Poisoned-Test \)) from the MNIST test set (\( MNIST-Test \)).

We create dataset \( P_0 \) by randomly selecting 1000 inputs from the clean MNIST test set (\( MNIST-Test \)). We consider the coverage obtained on this dataset as the baseline. From this baseline dataset (\( P_0 \)), we further create six different datasets by replacing \( \beta \% \) of clean inputs with poisoned inputs. We name these datasets as \( P_{\beta} \). For example, \( P_1 \) represents the dataset in which we replaced 1% (10) of the clean inputs with the same number of randomly selected poisoned inputs and \( P_{10} \) is the dataset in which we replaced 10% (100) of the clean inputs with the same number of randomly selected poisoned inputs. Similarly, we have \( P_2, P_3, P_5 \) and \( P_7 \). The size of each dataset is 1000.

For each dataset, we first calculate \( \Delta_{\beta} \) which is essentially the difference (similar to Eq. 3) between the coverage value of the metric on the baseline dataset (\( P_0 \)) and the respective dataset (\( P_{\beta} \)). \( \max(\Delta_{\beta}) \) is the maximum \( \Delta_{\beta} \) across datasets \( P_0–P_{10} \). \( \min(\Delta_{\beta}) \) is the minimum \( \Delta_{\beta} \) across datasets.

---

**Table 3** Accuracy of model on datasets

| Dataset       | MNIST (functional) | MNIST (adversarial) | MNIST (poisoned) |
|---------------|--------------------|---------------------|------------------|
| \( F_0/A_0/P_0 \) | N/A                | 98.22               | 98.52            |
| \( F_1/A_1/P_1 \) | 97.00              | 97.26               | 97.70            |
| \( F_2/A_2/P_2 \) | 97.48              | 96.26               | 96.74            |
| \( F_3/A_3/P_3 \) | 95.93              | 95.32               | 95.76            |
| \( F_5/A_5/P_5 \) | 96.30              | 93.42               | 93.96            |
| \( F_7/A_7/P_7 \) | 96.65              | 91.40               | 92.32            |
| \( F_{10}/A_{10}/P_{10} \) | 96.60            | 88.50               | 89.60            |

Column “Dataset” shows the name of the Datasets. Column “MNIST (Functional)” shows the Accuracy for Datasets \( F_1–F_{10} \). Column “MNIST (Adversarial)” shows the Accuracy for Datasets \( A_0–A_{10} \). Column “MNIST (Poisoned)” shows the Accuracy for Datasets \( P_0–P_{10} \).
| Scenario | Dataset | Metric | KMNC (10) | KMNC (1000) | TKNC (10) | TKNC (1000) | NBC | SNAC | NC (0) | NC (0.2) | NC (0.5) | NC (0.75) | MC/DC (SS) | MC/DC (SV) | MC/DC (VS) | MC/DC (VV) |
|----------|---------|--------|-----------|-------------|-----------|-------------|-----|------|--------|---------|---------|---------|----------|----------|----------|----------|----------|
| Functional | $F_1$ | Min | 0.00 | 0.00 | 0.00 | 0.00 | 2.80 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 216.20 |
| | Max | 254.60 | 16.00 | 245.60 | 245.60 | 0.80 | 0.80 | 251.00 | 251.40 | 253.40 | 250.40 | 0.00 | 0.00 | 47,426.40 | 80,757.40 |
| | Avg | 44.26 | 0.80 | 9.96 | 0.74 | 0.00 | 0.00 | 40.91 | 64.62 | 85.99 | 42.92 | 0.00 | 0.00 | 14,994.11 | 15,958.12 |
| | Std. | 67.43 | 1.67 | 28.09 | 4.80 | 0.02 | 0.03 | 41.07 | 67.64 | 82.06 | 63.53 | 0.00 | 0.00 | 14,461.88 | 18,205.57 |
| | Var. | 4,546.93 | 2.81 | 789.82 | 23.07 | 0.00 | 0.00 | 170.21 | 462.98 | 6738.52 | 40,563.26 | 0.00 | 0.00 | 217,047,915.68 | 343,441,740.25 |
| | $F_{10}$ | Min | 0.00 | 0.00 | 0.00 | 0.00 | 3.20 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 7,972.00 |
| | Max | 254.80 | 10.80 | 228.80 | 228.80 | 0.80 | 0.80 | 246.80 | 254.40 | 253.80 | 252.60 | 0.00 | 0.00 | 90,890.80 | 142,748.00 |
| | Avg | 72.01 | 0.80 | 11.23 | 0.75 | 0.00 | 0.00 | 39.53 | 80.42 | 110.70 | 80.85 | 0.00 | 0.00 | 44,167.25 | 40,287.12 |
| | Std. | 76.94 | 1.25 | 24.34 | 4.01 | 0.02 | 0.03 | 39.64 | 83.46 | 81.18 | 75.04 | 0.00 | 0.00 | 28,187.31 | 32,664.23 |
| | Var. | 5,919.58 | 1.55 | 592.63 | 16.11 | 0.00 | 0.00 | 1573.37 | 6967.05 | 6590.26 | 5630.99 | 0.00 | 0.00 | 795,678,292.52 | 1,069,718,510.41 |
| Adversarial$_{=0.30} | A_0 | Min | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| | Max | 255.00 | 12.60 | 243.80 | 246.40 | 0.80 | 0.80 | 232.00 | 246.00 | 252.20 | 244.80 | 0.00 | 0.00 | 98,840.40 | 125,897.80 |
| | Avg | 74.26 | 0.99 | 14.98 | 0.94 | 0.00 | 0.00 | 218.81 | 196.66 | 94.57 | 55.21 | 0.00 | 0.00 | 749.13 | 20,325.60 |
| | Std. | 78.73 | 1.42 | 33.36 | 6.42 | 0.02 | 0.03 | 40.92 | 61.28 | 73.75 | 71.70 | 0.00 | 0.00 | 7,946.83 | 40,408.32 |
| | Var. | 6,198.64 | 2.03 | 1112.89 | 41.19 | 0.00 | 0.00 | 1674.34 | 3755.88 | 5440.95 | 5143.82 | 0.00 | 0.00 | 63,197,042.79 | 2,019,902,359.70 |
| Poisoned | $P_0$ | Min | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 12,986.40 |
| | Max | 255.00 | 12.60 | 247.00 | 247.20 | 0.60 | 0.60 | 232.00 | 247.40 | 253.80 | 248.20 | 0.00 | 0.00 | 156,211.40 | 219,296.00 |
| | Avg | 65.99 | 1.00 | 15.84 | 0.95 | 0.00 | 0.00 | 221.60 | 199.85 | 122.49 | 99.68 | 0.00 | 0.00 | 882.18 | 66,915.34 |
| | Std. | 74.18 | 1.49 | 33.46 | 5.05 | 0.01 | 0.01 | 32.60 | 56.13 | 73.95 | 73.52 | 0.00 | 0.00 | 40,395.50 | 58,645.70 |
| | Var. | 5503.32 | 2.21 | 1120.00 | 25.46 | 0.00 | 0.00 | 1017.31 | 351.56 | 6076.93 | 5405.70 | 0.00 | 0.00 | 107,599,179.62 | 1,490,070,791.53 |
| | $P_{10}$ | Min | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 13,404.20 |
| | Max | 255.00 | 12.00 | 247.40 | 247.40 | 0.60 | 0.60 | 234.20 | 246.80 | 254.40 | 249.80 | 0.00 | 0.00 | 155,748.20 | 217,657.80 |
| | Avg | 68.33 | 1.00 | 15.74 | 0.95 | 0.01 | 0.01 | 221.95 | 199.37 | 118.89 | 95.75 | 0.00 | 0.00 | 878.16 | 67,644.00 |
| | Std. | 75.30 | 1.46 | 32.92 | 4.98 | 0.11 | 0.11 | 31.16 | 54.54 | 78.35 | 73.83 | 0.00 | 0.00 | 10,327.47 | 38,962.82 |
| | Var. | 5669.85 | 2.15 | 1084.01 | 24.84 | 0.00 | 0.00 | 971.28 | 2975.74 | 6139.18 | 5452.29 | 0.00 | 0.00 | 106,676,969.06 | 1,518,177,548.27 | 943,247,524.91 |
datasets $P_0$–$P_{10}$. We then calculate normalized difference ($\text{NCoverage}(P_\beta)$) in coverage (similar to Eq. 4).

We repeated similar experiments using poisoned data. Figure 5d summarizes the results. We can see that as the value of $\beta$ is increased from 0 to 10, five metrics (KMNC ($K$: 10), KMNC ($K$: 1000), TKNC ($K$: 1000), NBC and SNAC) report higher coverage (shown by the high normalized difference value). All variants of NC and MC/DC fail to differentiate between the datasets because they report almost similar coverage across all datasets. Four of the metrics (TKNC ($K$: 10), NC (Threshold: 0.50), MC/DC (SV) and MC/DC (VS)) even report lesser coverage on poisoned datasets. NC (Threshold: 0.00) and MC/DC (VV) achieve 100% coverage on all datasets (even when the dataset does not contain any poisoned inputs), and MC/DC (SS) obtains 0% coverage on all datasets (even when the dataset consists 10% of poisoned inputs). The results thus indicate that many of the considered metrics may not be adequate for checking the defect detection ability of a dataset.

4.4 Quantitative information

We also comment on the quantitative information that can be computed with DNNCov. We calculate minimum, maximum, average, standard deviation and variance of the number of inputs that cover each of the coverage obligations for each coverage criterion. The summarized results are shown in Table 4. We note that different test suites may have similar total coverage values but different coverage distributions. This cannot be highlighted by existing coverage metrics. For example, in Sect. 4.3.1, we mentioned that the value for MC/DC (VV) remained constant across all datasets ($A_0$ and $A_{10}$). However, if we analyze the corresponding quantitative information (see $A_0$ and $A_{10}$ rows in Table 4), we can see that even though the MC/DC (VV) remained 100% across $A_0$ and $A_{10}$ on average more input pairs (44,256 vs 35,184) covered each neuron pair on $A_{10}$ as compared to $A_0$. This indicates that dataset $A_{10}$ is better than dataset $A_0$ as it may reveal more defects in the neural network model.

4.5 Summary

Our results indicate that the considered metrics may not always be adequate for measuring the functional diversity and defect detection ability of a given test suite. DNNCov alleviates this concern by providing a unified framework for evaluating different metrics. The quantitative information computed by DNNCov can also be used for this purpose. To further address these limitations, we plan to develop new coverage metrics that are still in terms of the structure of the DNN, but also have semantic meaning, such as activation patterns [18].

5 Conclusion

In this article, we presented and evaluated several recently proposed structural coverage criteria for testing neural networks. We also described DNNCov, an integrated tool for helping developers with testing neural networks, by providing the means to measure, visualize and compare different, state-of-the-art testing criteria for neural networks. In the future, we plan to add more structural coverage criteria to the tool, as they appear in the related research. We are also working on a method for measuring KMNC, NBC, and SNAC coverage in the absence of the training set. One idea is to classify statistical outliers as boundary cases that would be part of NBC/SNAC and determine KMNC based on the non-boundary cases. Finally, we plan to improve the visualization component in DNNCov to display more information about the network architecture.

References

1. Pei, K., Cao, Y., Yang, J., Jana, S.: DeepXplore: automated white-box testing of deep learning systems. In: SOSP (2017)
2. Ma, L., Juefei-Xu, F., Zhang, F., Sun, J., Xue, M., Li, B., Chen, C., Su, T., Li, L., Liu, Y., et al.: DeepGauge: Multi-granularity testing criteria for deep learning systems. In: ASE (2018)
3. Sun, Y., Huang, X., Kroening, D., Sharp, J., Hill, M., Ashmore, R.: Structural test coverage criteria for deep neural networks. ACM Trans. Embed. Comput. Syst. 18, 1 (2019)
4. Lee, S., Cha, S., Lee, D., Oh, H.: Effective white-box testing of deep neural networks with adaptive neuron-selection strategy. In: ISSTA (2020)
5. Xie, X., Ma, L., Juefei-Xu, F., Xue, M., Chen, H., Liu, Y., Zhao, J., Li, B., Yin, J., See, S.: DeepHunter: A coverage-guided fuzz testing framework for deep neural networks. In: ISSTA (2019)
6. Sun, Y., Wu, M., Ruan, W., Huang, X., Kwiatkowska, M., Kroening, D.: Concolic testing for deep neural networks. In: ASE (2018)
7. Tian, Y., Zeng, Z., Wen, M., Liu, Y., Kuo, T.-y., Cheung, S.-C.: Evalnn: A toolbox for evaluating deep neural network models. In: 2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), pp. 45–48 (2020)
8. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: CAV (2017)
9. Kim, J., Feldt, R., Yoo, S.: Guiding deep learning system testing using surprise adequacy. In: ICSE (2019)
10. Deng, L.: The MNIST database of handwritten digit images for machine learning research. IEEE Signal Process. Mag. 29, 141 (2012)
11. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. In: Proceedings of the IEEE (1998)
12. LeCun, Y., Boser, B.E., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W.E., Jackel, L.D.: Handwritten digit recognition with a back-propagation network. In: NIPS (1989)
13. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)
14. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016)
15. Julian, K.D., Lee, R., Kochenderfer, M.J.: Validation of image-based neural network controllers through adaptive stress testing. In: ITSC (2020)

16. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199 (2013)

17. Gu, T., Liu, K., Dolan-Gavitt, B., Garg, S.: Badnets: evaluating backdooring attacks on deep neural networks. IEEE Access 7, 47230–47244 (2019). https://doi.org/10.1109/ACCESS.2019.2909068

18. Gopinath, D., Converse, H., Pasareanu, C.S., Taly, A.: Property inference for deep neural networks. In: 34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019, pp. 797–809. IEEE, (2019). https://doi.org/10.1109/ASE.2019.00079

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

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