Testing Deep Neural Networks

Youcheng Sun[1], Xiaowei Huang[2], Daniel Kroening[1], James Sharp[3], Matthew Hill[3], and Rob Ashmore[3]

[1] Department of Computer Science, University of Oxford, UK
[2] Department of Computer Science, University of Liverpool, UK
[3] Defence Science and Technology Laboratory (Dstl), UK

Abstract. Deep neural networks (DNNs) have a wide range of applications, and software employing them must be thoroughly tested, especially in safety-critical domains. However, traditional software test coverage metrics cannot be applied directly to DNNs. In this paper, inspired by the MC/DC coverage criterion, we propose a family of four novel test criteria that are tailored to structural features of DNNs and their semantics. We validate the criteria by demonstrating that the generated test inputs guided via our proposed coverage criteria are able to capture undesired behaviours in a DNN. Test cases are generated using a symbolic approach and a gradient-based heuristic search. By comparing them with existing methods, we show that our criteria achieve a balance between their ability to find bugs (proxied using adversarial examples) and the computational cost of test case generation. Our experiments are conducted on state-of-the-art DNNs obtained using popular open source datasets, including MNIST, CIFAR-10 and ImageNet.

Keywords: neural networks, test criteria, test case generation

1 Introduction

Artificial intelligence (AI), specifically deep neural networks (DNNs), can deliver human-level results in some specialist tasks. There is now a prospect of a wide-scale deployment of DNNs in safety-critical applications such as self-driving cars. This naturally raises the question how software implementing this technology should be tested, validated and ultimately certified to meet the requirements of the relevant safety standards [1].

Research and industrial communities worldwide are taking significant efforts towards the best practice for the safety assurance for learning-enabled autonomous systems. Among all efforts, we mention a few, including a proposal under consideration by IEEE to form an official technical committee for verification of autonomous systems [2], the Assuring Autonomy International Programme [3] which investigates the certification of learned models, etc. Moreover, as stated in [4], the machine learning algorithm should be verified with an appropriate level of coverage. This paper develops a technical solution to support these efforts.

The industry relies on testing as a primary means to provide stakeholders with information about the quality of the software product or service [5]. Research in software testing has resulted in a broad range of approaches to assess different software criticality levels (comprehensive reviews are given in e.g., [6,7,8]). In white-box testing, the structure of a program is exploited to (perhaps automatically) generate test cases. Code coverage criteria (or metrics) have been designed to guide the generation of test cases and evaluate the completeness of a test suite. For example, a test suite with 100% statement coverage exercises all statements at least once. While it is arguable the extent to which coverage ensures correct functionality, high coverage is able to increase users' confidence (or trust) in the program [6]. Structural coverage metrics are used as a means of assessment in several high-tier safety standards, which establish both statement and modified condition/decision coverage (MC/DC) are applicable measures. MC/DC was developed by NASA and has been widely adopted. It is used in avionics software development guidance to ensure adequate testing of applications with the highest criticality [9].

AI systems that use DNNs are typically implemented in software. However, (white-box) testing for traditional software cannot be directly applied to DNNs. In particular, the flow of control in DNNs is not sufficient to represent the knowledge that is learned during the training phase and thus it is not obvious how to define structural coverage criteria for DNNs [10]. Meanwhile, DNNs exhibit different ”bugs” from traditional software. Notably, *adversarial examples* [11], in which two apparently indistinguishable inputs cause contradicted decisions, are one of the most prominent safety concerns in DNNs.

We believe that the testing of DNNs, guided by proper coverage criteria, must help developers find bugs, quantify network robustness and analyse its internal structures. Also, developers can use the generated adversarial examples to re-train and improve the network. These enable developers to understand and compare different networks for any safety related argument.

Technically, DNNs contain not only an architecture, which bears some similarity with traditional software programs, but also a large set of parameters, which are tuned by the training procedure. Any approach to testing DNNs needs to consider the unique properties of DNNs, such as the syntactic connections between neurons in adjacent layers (neurons in a given layer interact with each other and then pass information to higher layers), the ReLU (Rectified Linear Unit) activation functions and the semantic relationship between layers.

In this paper, we propose a novel, white-box testing methodology for DNNs. In particular, we propose a family of four test criteria, inspired by the MC/DC test criterion [12] from traditional software testing, that fit the distinct properties of DNNs mentioned above. It is known that an overly weak criterion may lead to insufficient testing, e.g., 100% neuron coverage [13] can be achieved by a simple test suite comprised of few input vectors from the training dataset, and an overly strong criterion may lead to computational intractability, e.g., 100% safety coverage is shown in [14] as difficult to achieve (NP-hard). Our criteria, when applied to guide test case generation, can achieve both intensive testing (i.e., non-trivial to achieve 100% coverage) and computational feasibility. As a matter of fact, excepting the safety coverage criterion in [14], all existing structural test coverage criteria for DNNs [13,15] are special cases of our proposed criteria. Our criteria are the first work that is able to capture and quantify causal relations existing in a DNN that are critical for understanding the neural network behaviour [16,17].

Subsequently, we validate the utility of our MC/DC variant by applying it to different approaches to DNN testing. At first, we adopt state-of-the-art concolic testing for DNNs [18]. Concolic testing combines concrete testing and symbolic encoding of DNNs. Specifically, the linear programming (LP) based algorithm produces a new test case (i.e., an input vector) by encoding a fragment of the DNN and then optimises over an objective that is to minimise the difference between the new and the current input vector. LP can be solved efficiently in PTIME, so the concolic test case generation algorithms can generate a test suite with low computational cost for small to medium-sized DNNs. Meanwhile, we develop a gradient descent (GD) based algorithm that takes the test condition as the optimisation objective and searches for satisfiable test cases in an adaptive manner under the guidance of the first-order derivative of the DNNs, which is able to work with large-scale DNNs.

Finally, we experiment with our test coverage criteria on state-of-the-art neural networks of different sizes (from a few hundred up to millions of neurons) to demonstrate their utility with respect to four aspects: ① *bug finding* ② *DNN safety statistics* ③ *testing efficiency* ④ *DNN internal structure analysis*. Bugs here refer to adversarial examples.

2 Preliminaries: Deep Neural Networks

A (feedforward and deep) neural network, or DNN, is a tuple $\mathcal{N} = (L, T, \Phi)$, where $L = \{L_k \mid k \in \{1..K\}\}$ is a set of layers, $T \subseteq L \times L$ is a set of connections between layers and $\Phi = \{\phi_k \mid k \in \{2, \dots, K\}\}$ is a set of functions, one for each non-input layer. In a DNN, L_1 is the *input* layer, L_K is the *output* layer and layers other than input and output layers are called *hidden layers*. Each layer L_k consists of s_k *neurons* (or nodes). The l-th node of layer k is denoted by $n_{k,l}$. Each node $n_{k,l}$ for $1 < k < K$ and $1 \leq l \leq s_k$ is associated with two variables $u_{k,l}$ and $v_{k,l}$, to record its values before and after an activation function, respectively. The ReLU [19] is by far the most popular activation function for DNNs, according to which the *activation value* of each node of hidden layers is defined as

$$v_{k,l} = ReLU(u_{k,l}) = \begin{cases} u_{k,l} & \text{if } u_{k,l} \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Each input node $n_{1,l}$ for $1 \leq l \leq s_1$ is associated with a variable $v_{1,l}$ and each output node $n_{K,l}$ for $1 \leq l \leq s_K$ is associated with a variable $u_{K,l}$, because no activation function is applied on them. We let $D_{L_k} = \mathbb{R}^{s_k}$ be the vector space associated with layer L_k, one dimension for each variable $v_{k,l}$. Notably, every point $x \in D_{L_1}$ is an input.

Except for inputs, every node is connected to nodes in the preceding layer by pre-trained parameters such that for all k and l with $2 \leq k \leq K$ and $1 \leq l \leq s_k$, we have:

$$u_{k,l} = b_{k,l} + \sum_{1 \leq h \leq s_{k-1}} w_{k-1,h,l} \cdot v_{k-1,h} \tag{2}$$

where $w_{k-1,h,l}$ is the weight for the connection between $n_{k-1,h}$ (i.e., the h-th node of layer $k-1$) and $n_{k,l}$ (i.e., the l-th node of layer k), and $b_{k,l}$ is the so-called *bias* for node $n_{k,l}$. We note that this definition can express both fully-connected functions and convolutional functions. The function ϕ_k is the combination of Equations (1) and (2). Owing to the use of the ReLU as in (1), the behavior of a neural network is highly non-linear.

Finally, for any input, the DNN assigns a *label*, that is, the index of the node of output layer with the largest value: $label = \text{argmax}_{1 \le l \le s_K} u_{K,l}$. Let $\mathcal{L}$ be the set of labels.

Example 1. Figure 1 is a simple DNN with four layers. Its input space is $D_{L_1} = \mathbb{R}^2$ where $\mathbb{R}$ is the set of real numbers.

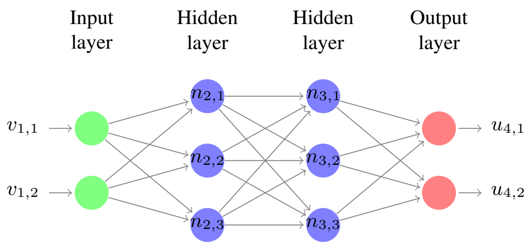


Fig. 1: A simple neural network

Given one particular input x, the DNN $\mathcal{N}$ is *instantiated* and we use $\mathcal{N}[x]$ to denote this instance of the network. In $\mathcal{N}[x]$, for each node $n_{k,l}$, the values of the variables $u_{k,l}$ and $v_{k,l}$ are fixed and denoted by $u_{k,l}[x]$ and $v_{k,l}[x]$, respectively. Therefore, the activation or deactivation of each ReLU operation in the network is also determined. We define

$$sign_{\mathcal{N}}(n_{k,l}, x) = \begin{cases} +1 & \text{if } u_{k,l}[x] = v_{k,l}[x] \\ -1 & \text{otherwise} \end{cases} \tag{3}$$

The subscript $\mathcal{N}$ will be omitted when clear from the context. The classification label of x is denoted by $\mathcal{N}[x].label$.

Example 2. Let $\mathcal{N}$ be a DNN whose architecture is given in Figure 1. Assume that the weights for the first three layers are given by $W_1 = \begin{bmatrix} 4 & 0 & -1 \\ 1 & -2 & 1 \end{bmatrix}$ and $W_2 = \begin{bmatrix} 2 & 3 & -1 \\ -7 & 6 & 4 \\ 1 & -5 & 9 \end{bmatrix}$ and that all biases are 0. When given an input $x = [0, 1]$, we get $sign(n_{2,1}, x) = +1$, since $u_{2,1}[x] = v_{2,1}[x] = 1$, and $sign(n_{2,2}, x) = -1$, since $u_{2,2}[x] = -2 \neq 0 = v_{2,2}[x]$.

We remark that, for simplicity of discussion, the definition focuses on DNNs with fully connected layers. However, as shown in our experiments, our method can also be applied to other popular DNN structures, such as convolutional and maxpooling layers, and sigmoid activation functions used in the state-of-the-art DNNs.

3 Adequacy Criteria for Testing DNNs

3.1 Test Coverage and MC/DC

A test adequacy criterion, or a test coverage metric, is used to quantify the degree of adequacy to which the software is tested by a test suite with respect to a set of test conditions. Throughout this paper, we use "criterion" and "metric" interchangeably.

Our criteria for DNNs are inspired by established practices in software testing, in particular MC/DC test criterion[12], but are designed for the specific attributes of DNNs. MC/DC is a method of measuring the extent to which safety-critical software has been adequately tested. At its core is the idea that if a choice can be made, all the possible factors (conditions) that contribute to that choice (decision) must be tested. For traditional software, both conditions and the decision are usually Boolean variables or Boolean expressions.

Example 3. The decision

$$d: \ ((a > 3) \vee (b = 0)) \wedge (c \neq 4) \tag{4}$$

contains the conditions: $(a > 3)$, $(b = 0)$ and $(c \neq 4)$. The following four test cases provide 100% MC/DC coverage:

1. $(a > 3)$=false, $(b = 0)$=true, $(c \neq 4)$=false
2. $(a > 3)$=true, $(b = 0)$=false, $(c \neq 4)$=true
3. $(a > 3)$=false, $(b = 0)$=false, $(c \neq 4)$=true
4. $(a > 3)$=false, $(b = 0)$=true, $(c \neq 4)$=true

The first two test cases already satisfy both *condition coverage* (i.e., all possibilities of the conditions are exploited) and *decision coverage* (i.e., all possibilities of the decision are exploited). The other two cases are needed because, for MC/DC, each condition should evaluate to true and false at least once, and should independently affect the decision outcome (e.g., the effect of the first condition can be seen by comparing cases 2 and 3).

3.2 Decisions and Conditions in DNNs

Our instantiation of the concepts "decision" and "condition" for DNNs is inspired by the similarity between Equation (2) and Equation (4) and the unique properties of DNNs. The information represented by nodes in the next layer can be seen as a summary (implemented by the layer function, the weights and the bias) of the information in the current layer. For example, it has been claimed that nodes in a deeper layer represent more complex attributes of the input [16,17].

We let $\Psi_k \subseteq \mathcal{P}(L_k)$ be a set of subsets of nodes at layer k. Without loss of generality, each element of Ψ_k, i.e., a subset of nodes in L_k, represents a *feature* learned at layer k. Therefore, *the core idea of our criteria is to ensure that not only the presence of a feature needs to be tested but also the effects of less complex features on a more complex feature must be tested.* We use $t_k = |\Psi_k|$ to denote the number of features in Ψ_k and $\psi_{k,l}$ for $1 \leq l \leq t_k$ the l-th feature. It is noted that the features can be overlapping, i.e., $\psi_{k,l_1} \cap \psi_{k,l_2} \neq \emptyset$. We consider every feature $\psi_{k,l}$ for $2 \leq k \leq K$ and $1 \leq l \leq t_k$ a *decision* and say that its *conditions* are those features connected to it in the layer $k-1$, i.e., $\{\psi_{k-1,l'} \mid 1 \leq l' \leq t_{k-1}\}$.

The use of feature generalises the basic building block in the DNN from a single node to a set of nodes. A single node can be represented as a singleton set. In practice, the feature can be supported by the tensor implementation in popular machine learning libraries [20] and various feature extraction methods such as SIFT [21], SURF [22], etc. To work with features, we extend the notations $u_{k,l}[x]$ and $v_{k,l}[x]$ for a node $n_{k,l}$ to a feature $\psi_{k,l}$ and write $\psi_{k,l}[x]$ and $\phi_{k,l}[x]$ for the vectors before and after ReLU, respectively.

Definition 1. *A feature pair $(\psi_{k,i}, \psi_{k+1,j})$ are two features in adjacent layers k and $k+1$ such that $1 \leq k < K$, $1 \leq i \leq t_k$ and $1 \leq j \leq t_{k+1}$. Given a DNN $\mathcal{N}$, we write $\mathcal{O}(\mathcal{N})$ (or, simply $\mathcal{O}$) for the set of its feature pairs. We may also call $(\psi_{k,i}, \psi_{k+1,j})$ a neuron pair when both $\psi_{k,i}$ and $\psi_{k+1,j}$ are singleton sets.*

Our new criteria are defined by capturing different ways of instantiating the changes of the conditions and the decision. Unlike Boolean variables or expressions, where it is trivial to define change, i.e., $\mathsf{true} \rightarrow \mathsf{false}$ or $\mathsf{false} \rightarrow \mathsf{true}$, in DNNs there are many different ways of defining that a decision is *affected* by the changes of the conditions. Before giving definitions for "affected" in Section 3.3, we start by clarifying when a feature "changes".

First, the change observed on a feature can be either a sign change or a value change.

Definition 2 (Sign Change). *Given a feature $\psi_{k,l}$ and two test cases x_1 and x_2, the sign change of $\psi_{k,l}$ is exploited by x_1 and x_2, denoted by $sc(\psi_{k,l}, x_1, x_2)$, iff*

– *$sign(n_{k,j}, x_1) \neq sign(n_{k,j}, x_2)$ for all $n_{k,j} \in \psi_{k,l}$.*

Moreover, we write $nsc(\psi_{k,l}, x_1, x_2)$ if

– *$sign(n_{k,j}, x_1) = sign(n_{k,j}, x_2)$ for all $n_{k,j} \in \psi_{k,l}$.*

Note that $nsc(\psi_{k,l}, x_1, x_2) \neq \neg sc(\psi_{k,l}, x_1, x_2)$. Before preceeding to another kind of change called *value change*, we need notation for value function. A value function is denoted by $g : \Psi_k \times D_{L_1} \times D_{L_1} \rightarrow \{\mathsf{true}, \mathsf{false}\}$. Simply speaking, it expresses the DNN developer's intuition (or knowledge) about what constitutes a significant change on the feature $\psi_{k,l}$, by specifying the difference between two vectors $\psi_{k,l}[x_1]$ and $\psi_{k,l}[x_2]$. We do not impose particular restrictions on the form of a value function, except that for practical reasons, it needs to be evaluated efficiently. Here, we give a few examples.

Example 4. For a singleton set $\psi_{k,l} = \{n_{k,j}\}$, the function $g(\psi_{k,l}, x_1, x_2)$ can express $|u_{k,j}[x_1] - u_{k,j}[x_2]| \geq d$ (absolute change) or $\frac{u_{k,j}[x_1]}{u_{k,j}[x_2]} > d \vee \frac{u_{k,j}[x_1]}{u_{k,j}[x_2]} < 1/d$ (relative change). It can also express the constraint on one of the values $u_{k,j}[x_2]$, such as $u_{k,j}[x_2] > d$ (upper boundary).

Example 5. For the general case, the function $g(\psi_{k,l}, x_1, x_2)$ can express the distance between two vectors $\psi_{k,l}[x_1]$ and $\psi_{k,l}[x_2]$ by norm-based distances $||\psi_{k,l}[x_1] - \psi_{k,l}[x_2]||_p \leq d$ for a real number d and a distance measure L^p, or structural similarity distances such as SSIM [23]. It can also express constraints between nodes of the same layer, such as $\bigwedge_{j \neq i} v_{k,i}[x_1] \geq v_{k,j}[x_1]$.

In general, the distance measure L^p could be L^1 (Manhattan distance), L^2 (Euclidean distance), L^∞ (Chebyshev distance) and so on. We remark that there is no consensus on which norm is the best to use and, furthermore, this is likely problem-specific. Finally, we define value change as follows.

Definition 3 (Value Change). *Given a feature $\psi_{k,l}$, two test cases x_1 and x_2, and a value function g, the value change of $\psi_{k,l}$ is exploited by x_1 and x_2 w.r.t. g, denoted by $vc(g, \psi_{k,l}, x_1, x_2)$, if*

- $g(\psi_{k,l}, x_1, x_2)$=*true.*

Moreover, we write $\neg vc(g, \psi_{k,l}, x_1, x_2)$ when the condition is not satisfied.

3.3 Covering Methods

In this section, we present a family of four methods to cover the causal changes in a DNN that were just defined.

Definition 4 (Sign-Sign Coverage, or SS Coverage). *A feature pair $\alpha = (\psi_{k,i}, \psi_{k+1,j})$ is SS-covered by two test cases x_1, x_2, denoted by $SS(\alpha, x_1, x_2)$, if the following conditions are satisfied by the DNN instances $\mathcal{N}[x_1]$ and $\mathcal{N}[x_2]$:*

- *$sc(\psi_{k,i}, x_1, x_2)$ and $nsc(P_k \setminus \psi_{k,i}, x_1, x_2)$;*
- *$sc(\psi_{k+1,j}, x_1, x_2)$.*

where P_k is the set of nodes in layer k.

SS coverage provides evidence that the sign change of a condition feature $\psi_{k,i}$ independently affects the sign of the decision feature $\psi_{k+1,j}$ of the next layer. Intuitively, the first condition says that the sign change of feature $\psi_{k,i}$ is exploited using x_1 and x_2, without changing the signs of other non-overlapping features. The second says that the sign change of feature $\psi_{k+1,j}$ is exploited using x_1 and x_2.

Example 6. (Continuation of Example 2) Given inputs $x_1 = (0.1, 0)$ and $x_2 = (0, -1)$, we compute the activation values for each node as given in Table 1. Therefore, we have $sc(\{n_{2,1}\}, x_1, x_2)$, $nsc(\{n_{2,2}\}, x_1, x_2)$, $nsc(\{n_{2,3}\}, x_1, x_2)$ and $sc(\{n_{3,1}\}, x_1, x_2)$. By Definition 4, the feature pair $(\{n_{2,1}\}, \{n_{3,1}\})$ is SS-covered by x_1 and x_2.

SS coverage is close to MC/DC: instead of observing the change of a Boolean variable (i.e., true $\rightarrow$ false or false $\rightarrow$ true), we observe a sign change of a feature. However, the behavior of a DNN has additional complexity that is not necessarily captured by a direct adoption of the MC/DC-style coverage to a DNN. **Subsequently, three additional coverage criteria are designed to complement SS coverage.**

First, the sign of $\psi_{k+1,j}$ can be altered between two test cases, even when none of the nodes $n_{k,i}$ in layer k changes its sign. Note that P_k, the set of all nodes in layer k, is also a feature and thus we write $nsc(P_k, x_1, x_2)$ to express that no sign change occurs for any of the nodes in layer k.

Definition 5 (Value-Sign Coverage, or VS Coverage). *Given a value function g, a feature pair $\alpha = (\psi_{k,i}, \psi_{k+1,j})$ is VS-covered by two test cases x_1, x_2, denoted by $VS^g(\alpha, x_1, x_2)$, if the following conditions are satisfied by the DNN instances $\mathcal{N}[x_1]$ and $\mathcal{N}[x_2]$:*

- *$vc(g, \psi_{k,i}, x_1, x_2)$ and $nsc(P_k, x_1, x_2)$;*

input	$n_{2,1}$	$n_{2,2}$	$n_{2,3}$	$n_{3,1}$	$n_{3,2}$	$n_{3,3}$
$(0.1, 0)$	0.4	0	-0.1 (0)	0.8	1.2	-0.4 (0)
$(0, -1)$	-1(0)	2	-1 (0)	-14 (0)	12	8
sign ch.	sc	¬sc	¬sc	sc	¬sc	sc
$(0, 1)$	1	-2 (0)	1	3	-2 (0)	8
$(0.1, 0.1)$	0.5	-0.2 (0)	0	1	1.5	-0.5 (0)
sign ch.	¬sc	¬sc	¬sc	¬sc	sc	sc
$(0, -1)$	-1 (0)	2	-1 (0)	-14 (0)	12	8
$(0.1, -0.1)$	0.3	0.2	-0.2 (0)	-0.8 (0)	2.1	0.5
sign ch.	sc	¬sc	¬sc	¬sc	¬sc	sc
$(0, 1)$	1	-2 (0)	1	3	-2 (0)	8
$(0.1, 0.5)$	0.9	-1 (0)	0.4	2.2	0.7	2.7
sign ch.	¬sc	¬sc	¬sc	¬sc	sc	¬sc

Table 1: Activation values and sign changes for the input examples in Examples 6, 7, 8, 9. An entry can be of the form v, in which $v \geq 0$, or $u(v)$, in which $u < 0$, $v = 0$, or sc, denoting that the sign has been changed, or $\neg sc$, denoting that there is no sign change.

– $sc(\psi_{k+1,j}, x_1, x_2)$.

Intuitively, the first condition describes the value change of nodes in layer k and the second requests the sign change of the feature $\psi_{k+1,j}$. Note that, in addition to $vc(g, \psi_{k,i}, x_1, x_2)$, we need $nsc(L_k, x_1, x_2)$, which asks for no sign changes for any node at layer k. This is to ensure that the overall change to the activations in layer k is relatively small.

Example 7. (Continuation of Example 2) Given two inputs $x_1 = (0, 1)$ and $x_2 = (0.1, 0.1)$, by the computed activation values in Table 1, we have $sc(\{n_{3,3}\}, x_1, x_2)$ and all nodes in layer 2 do not change their activation signs, i.e., $nsc(\{n_{2,1}, n_{2,2}, n_{2,3}\}, x_1, x_2)$. Thus, by Definition 5, x_1 and x_2 (subject to certain value function g) can be used to VS-cover the feature pair e.g., $(\{n_{2,1}, n_{2,2}\}, \{n_{3,3}\})$.

Until now, we have seen the sign change of a decision feature $\psi_{k+1,j}$ as the equivalent of the change of a decision in MC/DC. This view may still be limited. For DNNs, a key safety problem [11] related to their high non-linearity is that an insignificant (or imperceptible) change to the input (e.g., an image) may lead to a significant change to the output (e.g., its label). We expect that our criteria can guide test case generation algorithms towards unsafe cases, by working with two adjacent layers that are finer than the input-output relation. We notice that the label change in the output layer is the direct result of the changes to the activation values in the penultimate layer. Therefore, in addition to the sign change, the change of the value of the decision feature $\psi_{k+1,j}$ is also important.

Definition 6 (Sign-Value Coverage, or SV Coverage). *Given a value function g, a feature pair $\alpha = (\psi_{k,i}, \psi_{k+1,j})$ is SV-covered by two test cases x_1, x_2, denoted by $SV^g(\alpha, x_1, x_2)$, if the following conditions are satisfied by the DNN instances $\mathcal{N}[x_1]$ and $\mathcal{N}[x_2]$:*

– *$sc(\psi_{k,i}, x_1, x_2)$ and $nsc(P_k \setminus \psi_{k,i}, x_1, x_2)$;*

– $vc(g, \psi_{k+1,j}, x_1, x_2)$ *and* $nsc(\psi_{k+1,j}, x_1, x_2)$.

The first condition is the same as that in Definition 4. The difference is in the second condition, which now considers the feature value change $vc(g, \psi_{k+1,j}, x_1, x_2)$ with respect to a value function g, by independently modifying one its condition features' sign. Intuitively, SV Coverage captures the significant change of a decision feature's value that complements the sign change case.

Example 8. (Continuation of Example 2) Consider the feature pair $(\{n_{2,1}\}, \{n_{3,2}\})$. Given two inputs $x_1 = (0, -1)$ and $x_2 = (0.1, -0.1)$, by the computed activation values in Table 1, we have $sc(\{n_{2,1}\}, x_1, x_2)$ and $nsc(\{n_{2,2}, n_{2,3}\}, x_1, x_2)$. If, according to the function g, $\frac{u_{3,2}[x_1]}{u_{3,2}[x_2]} \approx 5.71$ is a significant change, i.e., $g(u_{3,2}[x_1], u_{3,2}\ [x_2])$=true, then the pair $(\{n_{2,1}\}, \{n_{3,2}\})$ is SV-covered by x_1 and x_2.

Finally, we have the following definition by replacing the sign change of the decision in Definition 5 with value change.

Definition 7 (Value-Value Coverage, or VV Coverage). *Given two value functions g_1 and g_2, a feature pair $\alpha = (\psi_{k,i}, \psi_{k+1,j})$ is VV-covered by two test cases x_1, x_2, denoted by $VV^{g_1,g_2}(\alpha, x_1, x_2)$, if the following conditions are satisfied by the DNN instances $\mathcal{N}[x_1]$ and $\mathcal{N}[x_2]$:*

– $vc(g_1, \psi_{k,i}, x_1, x_2)$ *and* $nsc(P_k, x_1, x_2)$;
– $vc(g_2, \psi_{k+1,j}, x_1, x_2)$ *and* $nsc(\psi_{k+1,j}, x_1, x_2)$.

Intuitively, VV coverage targets scenarios in which there is no sign change for a condition feature, but the decision feature's value is changed significantly.

Example 9. (Continuation of Example 2) For any $i \in \{1, \dots, 3\}$, the feature pair $(\{\psi_{2,i}\}, \{\psi_{3,3}\})$ are VV-covered by the inputs $x_1 = (0, 1)$ and $x_2 = (0.1, 0.5)$, subject to the value functions g_1 and g_2. As shown in Table 1, $\frac{u_{3,3}[x_1]}{u_{3,3}[x_2]} \approx 2.96$, for all $i \in \{1, \dots, 3\} : nsc(\{n_{2,i}\}, x_1, x_2)$ and $nsc(\{n_{3,3}\}, x_1, x_2)$.

3.4 Test Conditions, Test Suites and Test Criteria

By utilising the covering methods defined in Section 3.3, we now are able to instantiate the test conditions, test suites and test criteria for DNNs. Let $F = \{SS, VS^g, SV^g, VV^{g_1,g_2}\}$ be a set of covering methods. Given a DNN $\mathcal{N}$ and a covering method $f \in F$, a test condition set is characterised by the pair $(f, \mathcal{O})$ that asks for the coverage of corresponding causal changes on feature pairs in $\mathcal{O}$ according to f.

Given a DNN $\mathcal{N}$, a test suite $\mathcal{T}$ is a finite set of inputs, i.e., $\mathcal{T} \subseteq D_{L_1}$. Ideally, we run a test case generation algorithm to find a test suite $\mathcal{T}$ such that

$$\forall \alpha \in \mathcal{O}\ \exists\, x_1, x_2 \in \mathcal{T} : f(\alpha, x_1, x_2) \tag{5}$$

In practice, we might want to compute the degree to which the test conditions are satisfied by a generated test suite $\mathcal{T}$.

Definition 8 (Test Criterion). *Given a DNN $\mathcal{N}$, a test condition set by $(f, \mathcal{O})$ and a test suite $\mathcal{T}$, the test criterion $M_f(\mathcal{N}, \mathcal{T})$ is defined as follows:*

$$M_f(\mathcal{N}, \mathcal{T}) = \frac{|\{\alpha \in \mathcal{O} | \exists x_1, x_2 \in \mathcal{T} : f(\alpha, x_1, x_2)\}|}{|\mathcal{O}|} \tag{6}$$

That is, it computes the percentage of the feature pairs that are covered by test cases in $\mathcal{T}$ with respect to the covering method f.

Finally, instantiating f with covering methods in F, we obtain four test criteria $M_{SS}(\mathcal{N}, \mathcal{T})$, $M_{VS^g}(\mathcal{N}, \mathcal{T})$, $M_{SV^g}(\mathcal{N}, \mathcal{T})$ and $M_{VV^{g_1,g_2}}(\mathcal{N}, \mathcal{T})$.

4 Comparison with Existing Structural Test Criteria

So far, there have been a few proposals for structural test coverage criteria for DNNs. In this part, we compare our criteria with them, including the safety coverage (M_S) [14], neuron coverage (M_N) [13] and several of its extensions in [15] such as neuron boundary coverage (M_{NB}), multisection neuron coverage (M_{MN}) and top neuron coverage (M_{TN}). While [13] and [14] have been authored slightly ahead of ours, our criteria have been developed in parallel with [15].

A metric M_1 is said to be weaker than another metric M_2, denoted by $M_1 \preceq M_2$, iff for any given test suite $\mathcal{T}$ on $\mathcal{N}$, we have $M_1(\mathcal{N}, \mathcal{T}) < 1$ implies $M_2(\mathcal{N}, \mathcal{T}) < 1$. For instance, as shown in Example 3, decision coverage and condition coverage are weaker than MC/DC, since MC/DC cannot be covered before all decisions and conditions are covered.

The introduction of the feature relation in this work is very powerful: 1) the criteria in this paper are stronger than those in [13] and [15], which only consider individual neurons' activation statuses, and 2) it is non-trivial for the safety coverage in [14], which is comparable to the traditional path coverage that asks to cover every program execution path, to cover all test conditions of our criteria.

In the following, we uniformly formalise the criteria in [13,14,15] based on notations in this paper and we will define $M_f(\mathcal{N}, \mathcal{T})$ for $f \in \{N, S, NB, MN, TN\}$.

Definition 9 (Neuron Coverage). *A node $n_{k,i}$ is neuron covered by a test case x, denoted by $N(n_{k,i}, x)$, if $sign(n_{k,i}, x) = +1$.*

Given the definition, the neuron coverage asks that each neuron $n_{k,i}$ must be activated at least once by some test input x: $sign(n_{k,i}, x) = +1$.

The neuron coverage was later generalised in [15] to cover more fine-grained neuron activation statuses, including the boundary value for a neuron's activation. For simplicity, we only consider upper bounds when working with neuron boundary coverage. Given a node $n_{k,i}$ and a training dataset X, we let $v^u_{k,i} = \max_{x \in X} v_{k,i}[x]$ be its maximum value over the inputs in X.

Definition 10 (Neuron Boundary Coverage). *A node $n_{k,i}$ is neuron boundary covered by a test case x, denoted by $NB(n_{k,i}, x)$, if $v_{k,i}[x] > v^u_{k,i}$.*

Let $rank(n_{k,i}, x)$ be the rank of $v_{k,i}[x]$ among those values of the nodes at the same layer, i.e., $\{v_{k,j}[x] \mid 1 \leq j \leq s_k\}$.

Definition 11 (Top Neuron Coverage). *For $1 \leq m \leq s_k$, a node $n_{k,i}$ is top-m neuron covered by x, denoted by $TN^m(n_{k,i}, x)$, if $rank(n_{k,i}, x) \leq m$.*

Let $v^l_{k,i} = \min_{x \in X} v_{k,i}[x]$. We can split the interval $I_{k,i} = [v^l_{k,i}, v^u_{k,i}]$ into m equal sections and let $I^j_{k,i}$ be the jth section.

Definition 12 (Multisection Neuron Coverage). *Given $m \geq 1$, a node $n_{k,i}$ is m-multisection neuron covered by a test suite $\mathcal{T}$, denoted by $MN^m(n_{k,i}, \mathcal{T})$, if $\forall 1 \leq j \leq m \exists x \in \mathcal{T} : v_{k,i}[x] \in I^j_{k,i}$, i.e., all sections are covered by some test cases.*

Given $f \in \{N, NB, TN^m\}$ and the set $\mathcal{H}(\mathcal{N})$ of hidden nodes in $\mathcal{N}$, their associated test criterion can be then defined as follows

$$M_f(\mathcal{N}, \mathcal{T}) = \frac{|\{n \in \mathcal{H}(\mathcal{N}) \mid \exists x \in \mathcal{T} : f(n, x)\}|}{|\mathcal{H}(\mathcal{N})|} \tag{7}$$

$M_{MN^m}(\mathcal{N}, \mathcal{T})$ can be obtained by a simple adaptation.

We can fnd out that the criteria in [13,14] are special cases of our criteria (with a suitable value function g). As an example, the "weaker than" relationship between neuron coverage and SS coverage is proved in the lemma below.

Lemma 1. $M_N \preceq M_{SS}$.

Proof. Note that, for every hidden node $n_{k,j} \in \mathcal{H}(\mathcal{N})$, there exists a feature pair $(\{n_{k-1,i}\}, \{n_{k,j}\}) \in \mathcal{O}(\mathcal{N})$ for any $1 \leq i \leq s_{k-1}$. Then, by Definition 4, we have $sc(\{n_{k,j}\}, x_1, x_2)$, which by Definition 2 means that $sign(n_{k,j}, x_1) \neq sign(n_{k,j}, x_2)$. That is, either $sign(n_{k,j}, x_1) = +1$ or $sign(n_{k,j}, x_2) = +1$. Therefore, if $n_{k,j}$ is not covered in a test suite $\mathcal{T}_1$ for neuron coverage, none of the pairs $(\{n_{k-1,i}\}, \{n_{k,j}\})$ for $1 \leq i \leq s_{k-1}$ is covered in a test suite $\mathcal{T}_2$ for SS coverage.

Figure 2 gives a diagrammatic summary of the relations between all existing structural test coverage criteria for DNNs. The arrows represent the "weaker than" relations. The complete proofs are in the appendix. As shown in Figure 2, our criteria require more test cases to be generated than those in [13,15], and therefore can lead to more intensive testing.

On the other hand, as indicated in Figure 2, SS coverage is weaker than safety coverage [14]. In [14], the input space is discretised with a set of hyper-rectangles, and then one test case is generated for each hyper-rectangle. Such a scheme is computationally intractable due to the high-dimensionality of DNNs. The testing approach in this paper is more practical.

Definition 13 (Safety Coverage). *Let each hyper-rectangle rec contain those inputs with the same pattern of ReLU, i.e., for all $x_1, x_2 \in rec$ we have $sign(n_{k,l}, x_1) = sign(n_{k,l}, x_2)$ for all $n_{k,l} \in \mathcal{H}(\mathcal{N})$. A hyper-rectangle rec is covered by a test case x, denoted by $S(rec, x)$, if $x \in rec$.*

Let $Rec(\mathcal{N})$ be the set of hyper-rectangles. Then

$$M_S(\mathcal{N}, \mathcal{T}) = \frac{|\{rec \in Rec(\mathcal{N}) \mid \exists x \in \mathcal{T} : S(rec, x)\}|}{|Rec(\mathcal{N})|} \tag{8}$$

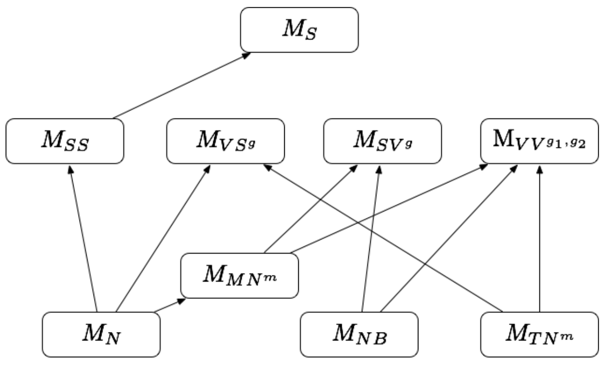


Fig. 2: Relationship between DNN structural test criteria

5 Automated Test Case Generation

We conjecture that the criteria proposed above achieve a good balance between their ability to guide test case generation towards relevant cases and computational cost. To show this hypothesis, we now apply our criteria with two different test case generation approaches for DNNs.

The test conditions required by our criteria exhibit particular combinations between the condition feature and the decision feature, and it is not trivial to generate test cases for them. Due to the lack of awareness of the feature relation, testing methods in [13,14,15] cannot be directly used to generate tests for our criteria. Also, as pointed out in [24], random test case generation is prohibitively inefficient for DNNs. Meanwhile, the symbolic encoding in the concolic testing method in [18] is expressive enough to encode test conditions defined by our criteria and is suitable for small to medium-sized DNNs. Furthermore, in this section, we also present a new test case generation algorithm based on gradient descent (GD) search, which scales to large DNNs.

5.1 Test Oracle

An oracle in software testing is a mechanism to detemine whether a test has passed or failed. The DNN $\mathcal{N}$ represents a function $\hat{\mathcal{F}}(x)$, which approximates $\mathcal{F}(x) : D_{L_1} \to \mathcal{L}$ that models perfect human perception capability. Therefore, the ultimate safety requirement is that for all test cases $x \in D_{L_1}$, we have $\hat{\mathcal{F}}(x) = \mathcal{F}(x)$. However, such a requirement is not practical because of the large number of inputs in D_{L_1} and the high cost of asking humans to label images. A pragmatic compromise, as done in many other works including [11,25], is to use the following oracle as a proxy.

Definition 14 (Oracle). *Given a finite set X of correctly labeled inputs, an input x' passes the oracle if there exists some $x \in X$ such that x and x' are close enough and $\hat{\mathcal{F}}(x') = \hat{\mathcal{F}}(x)$.*

Ideally, the question of whether two inputs x and x' are close enough is to be answered according to the human perception. In practice, this is approximated by various approaches, including norm-based distance measures. Specifically, given the norm L^p and an upper bound b for the distance, we say that two inputs x and x' are close iff $||x - x'||_p \leq b$. We write $close(x, x')$ for this relation. A pair of inputs that satisfies this definition are called *adversarial examples* if the label assigned to them by the DNN differs.

The choice of b is problem-specific. In our experiments, we evaluate the distribution of adversarial examples with respect to the distance (as illustrated in Figure 4 for one of the criteria). The use of this oracle focuses on adversarial examples in the DNN. There may exist other ways to define a test oracle for DNNs, and our criteria are independent from its particular definition.

5.2 Test Case Generation with LP

We first adopt the concolic testing approach in [18] to generate test cases that satisfy the test conditions defined by our criteria. In [18], test conditions are symbolically encoded using an linear programming (LP) model that is solved to obtain new test cases. Specifically, the LP-based approach fixes a particular pattern of node activations according to a given input x.

Though the overall behaviour of a DNN is highly non-linear, due to the use of e.g., the ReLU activation function, when the DNN is instantiated with a particular input, the activation pattern is fixed, and this corresponds to an LP model.

LP model of a DNN instance The variables used in the LP model are distinguished in **bold**. All variables are real-valued. Given an input x, the input variable $\mathbf{x}$, whose value is to be synthesized with LP, is required to have the identical activation pattern as x, i.e., $\forall n_{k,i} : sign(n_{k,i}, \mathbf{x}) = sign(n_{k,i}, x)$.

We use variables $\mathbf{u_{k,i}}$ and $\mathbf{v_{k,i}}$ to denote the values of a node $n_{k,i}$ before and after the application of ReLU, respectively. Then, we have the set $\mathcal{C}_1[x]$ of constraints to encode ReLU operations for a network instance, where $\mathcal{C}_1[x]$ is given as:

$$\begin{array}{l}\{\mathbf{u_{k,i}} \geq 0 \wedge \mathbf{v_{k,i}} = \mathbf{u_{k,i}} \mid sign(n_{k,i}, x) \geq 0, k \in [2, K), i \in [1 \dots s_k]\} \\ \cup\{\mathbf{u_{k,i}} < 0 \wedge \mathbf{v_{k,i}} = 0 \mid sign(n_{k,i}, x) < 0, k \in [2, K), i \in [1 \dots s_k]\}\end{array} \tag{9}$$

Note, the activation values $\mathbf{u}_{k,i}$ of each node is determined by the activation values $\mathbf{v}_{k-1,j}$ of those nodes in the prior layer. This is defined as in Equation (2). Therefore, we add the following set of constraints, $\mathcal{C}_2[x]$, as a symbolic encoding of nodes' activation values.

$$\{\mathbf{u}_{k,i} = \sum_{1 \leq j \leq s_{k-1}} \{w_{k-1,j,i} \cdot \mathbf{v}_{k-1,j}\} + b_{k,i} \mid k \in [2, K), i \in [1 \dots s_k]\} \tag{10}$$

The resulting LP model $\mathcal{C}[x] = \mathcal{C}_1[x] \cup \mathcal{C}_2[x]$ represents *a symbolic set of inputs* that have the identical activation pattern as x. Further, we can specify some optimisation objective *obj* and call an LP solver to find the optimal $\mathbf{x}$ (if one exists). In concolic testing, each time the DNN is instantiated with a concrete input x_1, the corresponding partial activation pattern serves as the base for the LP modeling, upon which a new test input x_2 may be found that satisfies the specified test condition.

5.3 Test Case Generation: a Heuristic Search

The LP optimisation in Section 5.2 provides a strong guarantee that is able to return an input pair as long as one exists. However, its scalability depends on the efficiency of LP solvers, and it is not trivial to apply such a testing method to large-scale DNNs with millions of neurons. In this part, we instead develop a heuristic algorithm based on gradient search. Note that, it has been widley shown that following gradient change is efficient in finding bugs in DNNs and has been utilised in existing DNN testing methods (eg., [13,15,26]). The algorithm, depicted in Algorithm 1, is used to find an input pair

Algorithm 1 $get\_input\_pair(f, \psi_{k,i}, \psi_{k+1,j})$

```
for each $x_1 \in data\_set$ do
  sample an input $x_2$ and a positive number $\epsilon$
  for a bounded number of steps do
    if $f((\psi_{k,i}, \psi_{k+1,j}), x_1, x_2)$ then return $x_1, x_2$
    update $\epsilon$
    if $\neg f^{widen}((\psi_{k,i}, \psi_{k+1,j}), x_1, x_2)$ then $x_2 \leftarrow x_2 - \epsilon \cdot \nabla\hat{\mathcal{F}}(x_2)$
    else $x_2 \leftarrow x_2 + \epsilon \cdot \nabla\hat{\mathcal{F}}(x_2)$
return None, None
```

x_1, x_2 such that the test condition of the covering method, f, over the feature pair, $\alpha = (\psi_{k,i}, \psi_{k+1,j})$, is satisfied; that is, $f(\alpha, x_1, x_2)$ is true. We use $f^{widen}(\alpha, x_1, x_2)$ for a widened version of the testing condition f, such that all its predicates on the features $\psi_{k,i}$ and $\psi_{k+1,j}$ are eliminated. It is supposed that x_1 is given, and intuitively starting from an input, x_2, if feature changes other than $\psi_{k,i}$ and $\psi_{k+1,j}$ do not meet the requirements of f, x_2 is moved closer to x_1, by following the gradient descent: $x_2 \leftarrow x_2 - \epsilon \cdot \nabla\hat{\mathcal{F}}(x_2)$, as an attempt to counteract such changes. This applies to the case when the activation sign changes on other condition features. Otherwise, the change between x_1 and x_2 can only exploit a subset of predicates (in the testing condition) from the given feature pair, and we update x_2 following the gradient ascent. The algorithm's gradient change follows an adaptive manner that comprises of a local search to update x_2 at each step, and a simple strategy for the overall search direction to move closer or further, with respect to x_1. In our implementation, we apply the FGSM (Fast Gradient Sign Method) [27] to initialise x_2 and ϵ, and use a binary search scheme to update ϵ at each step.

As a heuristic, the algorithm works when there exists two inputs x_1 and x_2 s.t. x_1 is from the given "data_set", x_2 is an input along the gradient search direction, and (x_1, x_2) satisfies the specified test condition.

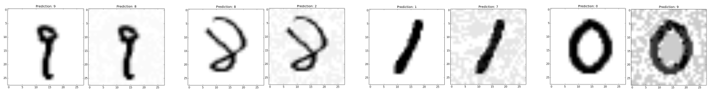
6 Experiments

We conduct experiments using the well-known MNIST Handwritten Image Dataset [28], the CIFAR-10 dataset [29] on small images and the ImageNet benchmark [30] from the large-scale visual recognition challenge. For clarity, our experiments are classified into four classes: ① *bug finding* ② *DNN safety statistics* ③ *testing efficiency* ④ *DNN* *internal structure analysis*, and results will be labeled correspondingly. We also explain the relation between our criteria and the existing ones.

In our implementation, the objective $\min ||x_2 - x_1||_\infty$ is used in all LP calls, to find good adversarial examples with respect to the test coverage conditions. Moreover, we use $g = \frac{u_{k+1,j}[x_2]}{u_{k+1,j}[x_1]} \geq \sigma$ with $\sigma = 2$ for g in SV^g and $\sigma = 5$ for VV^{g_1,g_2} (with respect to g_2). We admit that such choices are experimental. For generality and to speed up the experiments, we leave the value function g_1 unspecified. Providing a specific g_1 may require more effort to find an x_2 (because g_1 is an additional constraint), but the resulting x_2 can be better.

	hidden layers	M_{SS}	AE_{SS}	M_{VS^g}	AE_{VS^g}	M_{SV^g}	AE_{SV^g}	$M_{VV^{g_1,g_2}}$	$AE_{VV^{g_1,g_2}}$
$\mathcal{N}_1$	67x22x63	99.7%	18.9%	100%	15.8%	100%	6.7%	100%	21.1%
$\mathcal{N}_2$	59x94x56x45	98.5%	9.5%	100%	6.8%	99.9%	3.7%	100%	11.2%
$\mathcal{N}_3$	72x61x70x77	99.4%	7.1%	100%	5.0%	99.9%	3.7%	98.6%	11.0%
$\mathcal{N}_4$	65x99x87x23x31	98.4%	7.1%	100%	7.2%	99.8%	3.7%	98.4%	11.2%
$\mathcal{N}_5$	49x61x90x21x48	89.1%	11.4%	99.1%	9.6%	99.4%	4.9%	98.7%	9.1%
$\mathcal{N}_6$	97x83x32	100.0%	9.4%	100%	5.6%	100%	3.7%	100%	8.0%
$\mathcal{N}_7$	33x95x67x43x76	86.9%	8.8%	100%	7.2%	99.2%	3.8%	96%	12.0%
$\mathcal{N}_8$	78x62x73x47	99.8%	8.4%	100%	9.4%	100%	4.0%	100%	7.3%
$\mathcal{N}_9$	87x33x62	100.0%	12.0%	100%	10.5%	100%	5.0%	100%	6.7%
$\mathcal{N}_{10}$	76x55x74x98x75	86.7%	5.8%	100%	6.1%	98.3%	2.4%	93.9%	4.5%

Table 2: Coverage results on ten DNNs



(a) $9 \rightarrow 8$ (b) $8 \rightarrow 2$ (c) $1 \rightarrow 7$ (d) $0 \rightarrow 9$

Fig. 3: Selected adversarial examples for MNIST

6.1 MNIST

We randomly generate, and then train, a set of ten fully connected DNNs, such that each network has an accuracy of at least 97.0% on the MNIST validation data. The detailed network structure, and the number of neurons per layer, are given in Table 2. Every DNN input has been normalised into $[0, 1]^{28\times 28}$. Experiments were conducted on a MacBook Pro (2.5 GHz Intel Core i5, 8 GB memory).

We apply the covering method defined in Section 3. Besides the coverage M_f, we also measure the percentage of adversarial examples among all test pairs in the test suite, denoted by AE_f. *Thanks to the use of LP optimisation, the feature in this part is fine-grained to single neuron level.* That is, each feature pair is in fact a neuron pair.

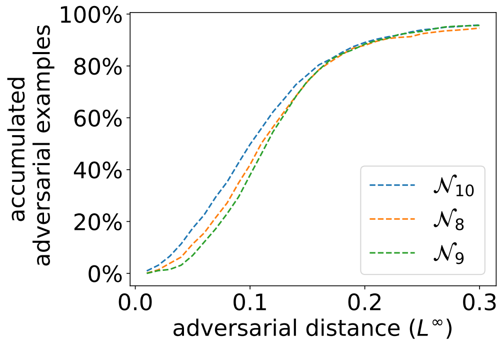
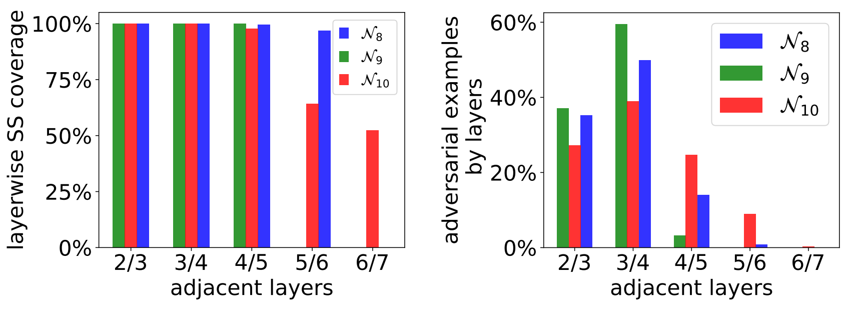


Fig. 4: Adversarial example curves that record the accumulated percentage of adversarial examples that fall into each distance: the adversarial distance measures the distance between an adversarial example and the original input



(a) (b)

Fig. 5: SS coverage by layer: (a) the coverage level per DNN layer; (b) the detected adversarial examples at each layer with respect to the total amount of adversarial examples

DNN Bug finding ① The testing results, as reported in Table 2, are promising: (1) the test case generation algorithm effectively achieves high coverage for all covering criteria, and (2) the covering methods are considered useful, supported by the fact that a significant portion of adversarial examples are identified. Figure 3 exhibits several adversarial examples found during the testing with different distances. We note that, *for neuron coverage [13], a high coverage can be easily achieved by selecting a few non-adversarial test cases that we generated.*

DNN safety analysis ② The coverage M_f and adversarial example percentage AE_f together provide quantitative statistics to evaluate a DNN. Generally speaking, given a test suite, a DNN with a high coverage level M_f and a low adversarial percentage AE_f is considered robust. In addition, we can study the *adversarial quality* by plotting a distance curve to see how close the adversarial example is to the correct input. Take a closer look into the results of SS coverage for the last three DNNs in Table 2. As illustrated in Figure 4, the horizontal axis measures the L^∞ distance and the vertical axis reports the accumulated percentage of adversarial examples that fall into this distance. A more robust DNN will have its shape in the small distance end (towards 0) lower, as the reported adversarial examples are relatively farther from their original correct inputs. Intuitively, this means that more effort needs to be made to fool a robust DNN from correct classification into mislabelling.

Layerwise behavior ④ Our experiments show that different layers of a DNN exhibit different behaviors in testing. Figure 5 reports the SS coverage results, collected in adjacent layers. In particular, Figure 5a gives the percentage of covered neuron pairs within individual adjacent layers. As we can see, when going deeper into the DNN, it can become harder to cover of neuron pairs. Under such circumstances, to improve the coverage performance, the use of larger a $data\_set$ is needed when generating test pairs. Figure 5b gives the percentage of adversarial examples found at different layers (among the overall adversarial examples). Interestingly, it seems that most adversarial examples are found when testing the middle layers.

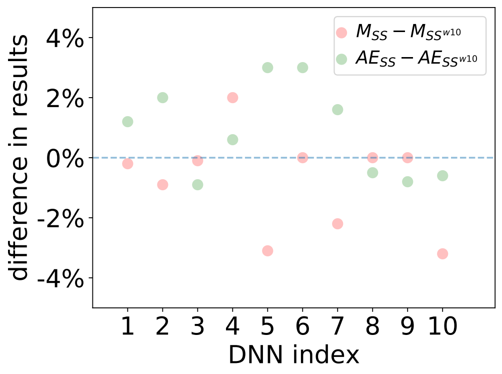


Fig. 6: SS vs. SS^{w10}. Results demonstrate that the SS coverage and its top-weight simplification have similar coverage levels ($M_{SS} - M_{SS^{w10}}$) and percentages of adversarial examples ($AE_{SS} - AE_{SS^{w10}}$)

SS coverage with top weights ③ For SS coverage criteria with neuron pairs, there are totally $|O|$ test conditions for $O \subseteq \mathcal{O}(\mathcal{N})$. We note that $|\mathcal{O}(\mathcal{N})| = \sum_{k=2}^{K} s_k \cdot s_{k-1}$. To reduce the test suite size, we define O as follows: $(\psi_{k,i}, \psi_{k+1,j}) \in O$ only when the weight is one of the κ largest among $\{|w_{k,i',j}| \mid i' \in [1 \dots s_k]\}$. The rationale is that condition neurons do not equally affect their decision, and those with higher (absolute) weights are likely to have a larger influence.

Figure 6 shows the difference, on coverage and adversarial example percentages, between SS coverage and its simplification with $\kappa = 10$, denoted by SS^{w10}. In general, the two are comparable. This is very useful in practice, as the "top weights" simplification mitigates the size of the rsulting test suite, and it is thus able to behave as a faster pre-processing phase and even provide an alternative with comparable results for SS coverage.

Cost of LP call ③ Since LP encoding of the DNN (partial) activation pattern plays a key role in the test generation, in this part we give details of the LP call cost, even though LP is widely accepted as an efficient method. For every DNN, we select a set of neuron pairs, where each decision neuron is at a different layer. Then, we measure the number of variables and constraints, and the time t in seconds (averaged over 100 runs) spent on solving each LP call. Results in Table 3 confirm that the LP model of a partial activation pattern is indeed lightweight, and its complexity increases in a linear manner when traversing into deeper layers of a DNN.

	$\mathcal{N}_8$			$\mathcal{N}_9$			$\mathcal{N}_{10}$		
	#vars	$\|\mathcal{C}\|$	t	#vars	$\|\mathcal{C}\|$	t	#vars	$\|\mathcal{C}\|$	t
L2-3	864	3294	0.58	873	3312	0.57	862	3290	0.49
L3-4	926	3418	0.84	906	3378	0.61	917	3400	0.71
L4-5	999	3564	0.87	968	3502	0.86	991	3548	0.75
L5-6	1046	3658	0.91	–	–	–	1089	3744	0.82
L6-7	–	–	–	–	–	–	1164	3894	0.94

Table 3: Number of variables and constraints, and time cost of each LP call in test generation

6.2 CIFAR-10

The CIFAR-10 dataset is a collection of 32x32 color images in ten kinds of objects. Different from the MNIST case, we need to train a DNN with convolutional layers in order to handle the CIFAR-10 image classification problem. Without loss of generality, the activation of a node in the convolutional layer is computed by the activations of a subset of precedent nodes, and each node belongs to a *feature map* in its layer. We apply the test case generation in Algorithm 1 for the SS coverage and measure the coverage results individually for decision features at each different layer. Overall, an SS coverage higher than 90% is achieved with a significant portion of adversarial examples. An interesting observation is as in Figure 7 (④), which shows that in this case the causal changes of features at deeper layers are able to detect smaller perturbations of inputs that cause adversarial behaviours, and this is likely to provide helpful feedback for developers to debug or tune the neural network parameters. Selected adversarial examples are given in Figure 8.

6.3 ImageNet

We applied our methods to VGG16 [31], a large-scale DNN trained on the ImageNet dataset. The heuristic search Algorithm 1 is called to generate test cases. We consider

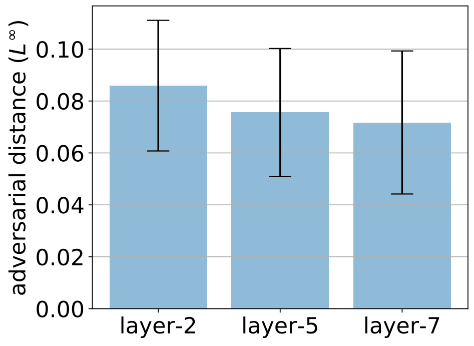
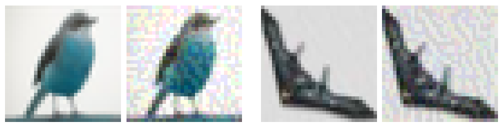


Fig. 7: The averaged adversarial distance for decision features at different layers



(a) bird $\rightarrow$ airplane (b) airplane $\rightarrow$ cat

Fig. 8: Selected adversarial examples for CIFAR-10

each decision feature as a single set of neurons. While we can use feature extraction methods such as SIFT [21] to obtain condition features, in our experiments we consider each condition feature as an arbitrary set of neurons for better exploration of the testing method. In particular, a size parameter ω is defined for the experiments such that a feature $\psi_{k,i}$ is required to have its size $\leq \omega \cdot s_k$. Recall that s_k is the number of neurons in layer k.

Different feature sizes ① ② ④ We apply SS coverage on 2,000 randomly sampled feature pairs with $\omega \in \{0.1\%, 0.5\%, 1.0\%\}$. The covering method shows its effectiveness by returning a test suite in which 10.5%, 13.6% and 14.6% are adversarial examples. We report the adversarial examples' average distance and standard deviation in Figure 9. The results confirm that there is a relation between the feature pairs and the input perturbation. Among the generated adversarial examples, a more fine-grained feature is able to capture smaller perturbations than a coarse one.

Results in Figure 9 are measured with L^∞-norm that corresponds to the maximum changes to a pixel. We observed that, though the change of each pixel is very small, for every adversarial example a large portion (around 50%) of pixels are changed. A typical adversarial example image is given in Figure 10. Overall, the detected adversarial examples are considered of high quality.

SV with neuron boundary coverage ① ② As shown in Section 4, our covering methods are stronger than neuron boundary coverage. In fact, neuron boundary is a special case of SV coverage, when the value function of the decision feature is designed to make the activation exceed the specified boundary value. We also validated this relation in

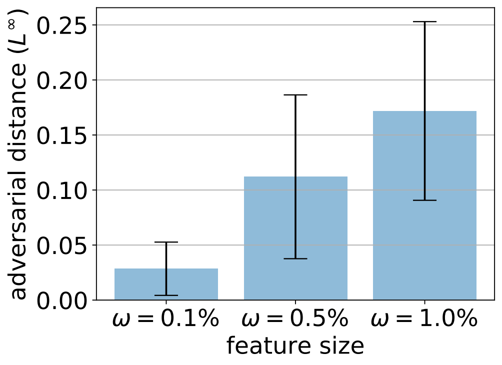


Fig. 9: Adversarial distance with different feature sizes: a smaller distance corresponds to more subtle adversarial examples

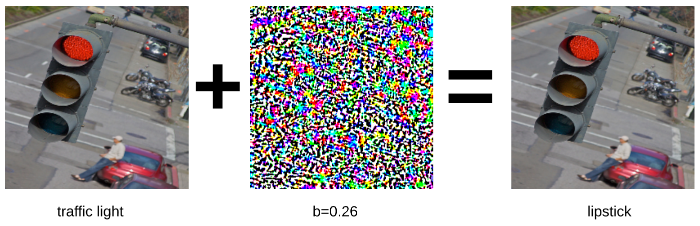


Fig. 10: An adversarial example ("lipstick") for the original traffic light input

the empirical manner, similarly to the experiments above, by generating a test suite using SV with neuron boundary coverage. We noticed that accessing boundary activation values is likely to request bigger changes to be made in DNNs. We set the feature size using $\omega = 10\%$ and obtain a test suite with 22.7% adversarial examples. However, the distance of these adversarial examples, with average L^∞-norm distance 3.49 and standard deviation 3.88, is much greater than those for the SS coverage, as in Figure 9.

7 Related Work

In the following, we briefly discuss existing techniques looking to validate safety properties of DNNs.

Generation of Adversarial Examples for DNNs Most existing work, e.g., [11,27,32,33] applies various heuristic algorithms, generally using search algorithms based on gradient descent or evolutionary techniques. These approaches may be able to find adversarial examples efficiently, but are *not able to provide any guarantee* (akin to verification) or *any certain level of confidence* (akin to testing) about the nonexistence of adversarial examples when the algorithm fails to find one.

Testing of DNNs At present, there are only a few proposals for structural DNN test coverage criteria. In [13], neuron coverage is proposed to cover each neuron's binary activation statuses. It is applied in [34] to guide the testing of DNN-driven autonomous cars. Extensions of neuron coverage are made in [15], which include a set of test criteria to check the corner values of a neuron's activation and the activation levels of a subset of neurons in the same layer. However, criteria in [13,15] simply ignore the key causal relationship in a DNN. Odena and Goodfellow [24] apply the approximate nearest neighbors algorithm to guide their tests generation, but it is not clear, in a DNN, what the maximum number of nearest neighbors are. As shown in [35], quantitative DNN coverage criteria can be applied to the design and certification of automotive systems with deep learning components.

In [14], the input space is discretised with hyper-rectangles, and then one test case is generated for each hyper-rectangle. The resulting safety coverage is a strong criterion, but the generation of a test suite can be very expensive. Whilst in [36], coverage is enforced to finite partitions of the input space, relying on predefined sets of application-specific scenario attributes. The ”boxing clever” technique in [37] focuses on the distribution of training data and divides the input domain into a series of representative boxes. In [38], the difference between test dataset and training dataset is measured by quantifying the difference between DNNs' activation patterns.

Some traditional test case generation techniques such as concolic testing [18,39], symbolic execution [40] and fuzzing [24,41] have been recently extended to DNNs. Mutation testing has similarly been investigated in [42,43,44,45,46]. And metamorphic testing [47,48,49] has been identified as a suitable test oracle for the robustness problem. The combinatorial method is explored to reduce the testing space for DNNs in [50]. Multi-implementation testing is applied to k-Nearest Neighbor (kNN) and Naive Bayes supervised learning algorithms in [51]. In [52], the adversarial inputs are treated as the fairness problem via testing.

Tensorflow [20] is a popular library for developing deep learning models, and Zhang et al [53] studied a collection of 175 bugs in Tensorflow programs. A testing framework is developed in [54] for learning based malware detection applications in Android. Autonomous driving is the primary application domain for assessments of DNN testing techniques [26,55,56].

Automated Verification of DNNs The safety problem of a DNN can be reduced into a constraint solving problem [57]. SMT [25,58,59,60], MILP [61,62,63,64,65] and SAT [66,67] solutions have already been considered. In [68], the DNN is transformed into into an equivalent hybrid system. These approaches typically only work with small networks with a few hundred hidden neurons, and approximation techniques [69,70,71,72,73,74,75] can be applied to improve the efficiency. Another thread of work [76,77,78] based on global optimisation is promising to work with larger networks.

8 Conclusions

We have proposed a set of novel test criteria for DNNs. Our experiments on various datasets and test case generation methods show promising results, indicating the feasibility and effectiveness of the proposed test criteria. The test coverage metrics developed within this paper provide a method to obtain evidence towards adversarial robustness, which is envisaged to contribute to safety cases. The metrics are also expected to provide additional insights for domain experts when they are considering the adequacy of a particular dataset for use in an application.

Acknowledgements This document is an overview of UK MOD (part) sponsored research and is released for informational purposes only. The contents of this document should not be interpreted as representing the views of the UK MOD, nor should it be assumed that they reflect any current or future UK MOD policy. The information contained in this document cannot supersede any statutory or contractual requirements or liabilities and is offered without prejudice or commitment.

References

1. Xiaowei Huang et al. Safety and trustworthiness of deep neural networks: A survey. *arXiv preprint arXiv:1812.08342*, 2018.
2. Verification of autonomous systems. `https://www.robotistry.org/vaswg/index.html`.
3. Assuring autonomy international programme. `https://www.york.ac.uk/assuring-autonomy/`.
4. SASWG. Safety assurance objectives for autonomous systems. 2019.
5. Cem Kaner. Exploratory testing. In *Quality Assurance Institute Worldwide Annual Software Testing Conference*, 2006.
6. Hong Zhu, Patrick AV Hall, and John HR May. Software unit test coverage and adequacy. *ACM Computing Surveys*, 29(4):366–427, 1997.
7. Yue Jia and Mark Harman. An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*, 37(5):649–678, 2011.
8. Ting Su, Ke Wu, Weikai Miao, Geguang Pu, Jifeng He, Yuting Chen, and Zhendong Su. A survey on data-flow testing. *ACM Computing Surveys*, 50(1):5:1–5:35, March 2017.
9. RTCA. DO-178C, software considerations in airborne systems and equipment certification. 2011.
10. Rob Ashmore and Elizabeth Lennon. Progress towards the assurance of non-traditional software. In *Safety-critical Systems Symposium*, 2017.
11. Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *In ICLR*. Citeseer, 2014.
12. Kelly Hayhurst, Dan Veerhusen, John Chilenski, and Leanna Rierson. A practical tutorial on modified condition/decision coverage. Technical report, NASA, 2001.
13. Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. DeepXplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 1–18. ACM, 2017.

14. Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. Feature-guided black-box safety testing of deep neural networks. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2018.
15. Lei Ma, Felix Juefei-Xu, Jiyuan Sun, Chunyang Chen, Ting Su, Fuyuan Zhang, Minhui Xue, Bo Li, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang. DeepGauge: Comprehensive and multi-granularity testing criteria for gauging the robustness of deep learning systems. In *Automated Software Engineering (ASE), 33rd IEEE/ACM International Conference on*, 2018.
16. Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.
17. Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 2018.
18. Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. Concolic testing for deep neural networks. In *Automated Software Engineering (ASE), 33rd IEEE/ACM International Conference on*, 2018.
19. Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pages 807–814, 2010.
20. Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. TensorFlow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.
21. David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004.
22. Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (SURF). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008.
23. Zhou Wang, Eero P Simoncelli, and Alan C Bovik. Multiscale structural similarity for image quality assessment. In *Signals, Systems and Computers, Conference Record of the Thirty-Seventh Asilomar Conference on*, 2003.
24. Augustus Odena and Ian Goodfellow. TensorFuzz: Debugging neural networks with coverage-guided fuzzing. *arXiv preprint arXiv:1807.10875*, 2018.
25. Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification*, pages 3–29. Springer, 2017.
26. Shakiba Yaghoubi and Georgios Fainekos. Gray-box adversarial testing for control systems with machine learning components. In *International Conference on Hybrid Systems: Computation and Control*, pages 179–184. ACM, 2019.
27. Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.
28. Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
29. Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.
30. Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.
31. VGG16 model for Keras. https://gist.github.com/baraldilorenzo/07d7802847aaad0a35d3.
32. Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), IEEE European Symposium on*, pages 372–387, 2016.
33. Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *Security and Privacy (S&P), IEEE Symposium on*, pages 39–57, 2017.

34. Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. DeepTest: Automated testing of deep-neural-network-driven autonomous cars. *arXiv preprint arXiv:1708.08559*, 2017.
35. Shuyue Lan, Chao Huang, Zhilu Wang, Hengyi Liang, Wenhao Su, and Qi Zhu. Design automation for intelligent automotive systems. In *International Test Conference (ITC)*, pages 1–10. IEEE, 2018.
36. Chih-Hong Cheng, Chung-Hao Huang, and Hirotoshi Yasuoka. Quantitative projection coverage for testing ML-enabled autonomous systems. In *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2018.
37. Rob Ashmore and Matthew Hill. Boxing clever: Practical techniques for gaining insights into training data and monitoring distribution shift. In *First International Workshop on Artificial Intelligence Safety Engineering*, 2018.
38. Jinhan Kim, Robert Feldt, and Shin Yoo. Guiding deep learning system testing using surprise adequacy. In *International Conference on Software Engineering*. IEEE, 2019.
39. Youcheng Sun, Xiaowei Huang, Daniel Kroening, James Shap, Matthew Hill, and Rob Ashmore. DeepConcolic: testing and debugging deep neural networks. In *International Conference on Software Engineering: Companion*. IEEE, 2019.
40. Divya Gopinath, Kaiyuan Wang, Mengshi Zhang, Corina S Pasareanu, and Sarfraz Khurshid. Symbolic execution for deep neural networks. *arXiv preprint arXiv:1807.10439*, 2018.
41. Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Hongxu Chen, Minhui Xue, Bo Li, Yang Liu, Jianjun Zhao, Jianxiong Yin, and Simon See. Coverage-guided fuzzing for deep neural networks. *arXiv preprint arXiv:1809.01266*, 2018.
42. Jingyi Wang, Guoliang Dong, Jun Sun, Xinyu Wang, and Peixin Zhang. Adversarial sample detection for deep neural network through model mutation testing. In *Proceedings of the 41st International Conference on Software Engineering*. IEEE, 2019.
43. Jingyi Wang, Jun Sun, Peixin Zhang, and Xinyu Wang. Detecting adversarial samples for deep neural networks through mutation testing. *arXiv preprint arXiv:1805.05010*, 2018.
44. Lei Ma, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Felix Juefei-Xu, Chao Xie, Li Li, Yang Liu, Jianjun Zhao, et al. DeepMutation: Mutation testing of deep learning systems. In *Software Reliability Engineering, IEEE 29th International Symposium on*, 2018.
45. Dawei Cheng, Chun Cao, Chang Xu, and Xiaoxing Ma. Manifesting bugs in machine learning code: An explorative study with mutation testing. In *International Conference on Software Quality, Reliability and Security (QRS)*, pages 313–324. IEEE, 2018.
46. Weijun Shen, Jun Wan, and Zhenyu Chen. MuNN: Mutation analysis of neural networks. In *International Conference on Software Quality, Reliability and Security Companion, QRS-C*. IEEE, 2018.
47. Junhua Ding, Xiaojun Kang, and Xin-Hua Hu. Validating a deep learning framework by metamorphic testing. In *Metamorphic Testing (MET), 2017 IEEE/ACM 2nd International Workshop on*, pages 28–34. IEEE, 2017.
48. Anurag Dwarakanath, Manish Ahuja, Samarth Sikand, Raghotham M Rao, RP Bose, Neville Dubash, and Sanjay Podder. Identifying implementation bugs in machine learning based image classifiers using metamorphic testing. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 118–128. ACM, 2018.
49. Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. Deep-Road: GAN-based metamorphic autonomous driving system testing. In *Automated Software Engineering (ASE), 33rd IEEE/ACM International Conference on*, 2018.
50. Lei Ma, Fuyuan Zhang, Minhui Xue, Bo Li, Yang Liu, Jianjun Zhao, and Yadong Wang. Combinatorial testing for deep learning systems. *arXiv preprint arXiv:1806.07723*, 2018.
51. Siwakorn Srisakaokul, Zhengkai Wu, Angello Astorga, Oreoluwa Alebiosu, and Tao Xie. Multiple-implementation testing of supervised learning software. In *Proc. AAAI-18 Workshop on Engineering Dependable and Secure Machine Learning Systems (EDSMLS)*, 2018.

52. Sakshi Udeshi, Pryanshu Arora, and Sudipta Chattopadhyay. Automated directed fairness testing. In *Automated Software Engineering (ASE), 33rd IEEE/ACM International Conference on*, 2018.
53. Yuhao Zhang, Yifan Chen, Shing-Chi Cheung, Yingfei Xiong, and Lu Zhang. An empirical study on TensorFlow program bugs. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2018.
54. Wei Yang and Tao Xie. Telemade: A testing framework for learning-based malware detection systems. In *Proc. AAAI-18 Workshop on Engineering Dependable and Secure Machine Learning Systems (EDSMLS)*, 2018.
55. Tommaso Dreossi, Shromona Ghosh, Alberto Sangiovanni-Vincentelli, and Sanjit A Seshia. Systematic testing of convolutional neural networks for autonomous driving. *arXiv preprint arXiv:1708.03309*, 2017.
56. Cumhur Erkan Tuncali, Georgios Fainekos, Hisahiro Ito, and James Kapinski. Simulation-based adversarial test generation for autonomous vehicles with machine learning components. In *Intelligent Vehicles Symposium (IV)*, pages 1555–1562. IEEE, 2018.
57. Tommaso Dreossi, Shromona Ghosh, Alberto Sangiovanni-Vincentelli, and Sanjit A Seshia. A formalization of robustness for deep neural networks. *arXiv preprint arXiv:1903.10033*, 2019.
58. Luca Pulina and Armando Tacchella. An abstraction-refinement approach to verification of artificial neural networks. In *International Conference on Computer Aided Verification*, pages 243–257. Springer, 2010.
59. Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.
60. Cumhur Erkan Tuncali, Hisahiro Ito, James Kapinski, and Jyotirmoy V Deshmukh. Reasoning about safety of learning-enabled components in autonomous cyber-physical systems. In *55th Design Automation Conference (DAC)*, pages 1–6. IEEE, 2018.
61. Rudy Bunel, Ilker Turkaslan, Philip HS Torr, Pushmeet Kohli, and M Pawan Kumar. Piecewise linear neural network verification: A comparative study. *arXiv preprint arXiv:1711.00455*, 2017.
62. Alessio Lomuscio and Lalit Maganti. An approach to reachability analysis for feed-forward ReLU neural networks. *arXiv preprint arXiv:1706.07351*, 2017.
63. Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. Maximum resilience of artificial neural networks. In Deepak D'Souza and K. Narayan Kumar, editors, *Automated Technology for Verification and Analysis*, pages 251–268. Springer, 2017.
64. Weiming Xiang, Hoang-Dung Tran, and Taylor T Johnson. Output reachable set estimation and verification for multi-layer neural networks. *arXiv preprint arXiv:1708.03322*, 2017.
65. Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Output range analysis for deep feedforward neural networks. In *NASA Formal Methods Symposium*, pages 121–138. Springer, 2018.
66. Nina Narodytska, Shiva Prasad Kasiviswanathan, Leonid Ryzhyk, Mooly Sagiv, and Toby Walsh. Verifying properties of binarized deep neural networks. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
67. Nina Narodytska. Formal analysis of deep binarized neural networks. In *IJCAI*, pages 5692–5696, 2018.
68. Radoslav Ivanov, James Weimer, Rajeev Alur, George J. Pappas, and Insup Lee. Verisig: Verifying safety properties of hybrid systems with neural network controllers. In *International Conference on Hybrid Systems: Computation and Control*, pages 169–178. ACM, 2019.
69. Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning*, pages 3575–3583, 2018.

70. Tommaso Dreossi, Alexandre Donzé, and Sanjit A Seshia. Compositional falsification of cyber-physical systems with machine learning components. *Journal of Automated Reasoning*, 2018.
71. Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, pages 5283–5292, 2018.
72. Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. In *USENIX Security Symposium*. USENIX Association, 2018.
73. Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. AI2: Safety and robustness certification of neural networks with abstract interpretation. In *Security and Privacy (SP), 2018 IEEE Symposium on*, 2018.
74. Souradeep Dutta, Xin Chen, and Sriram Sankaranarayanan. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *International Conference on Hybrid Systems: Computation and Control*, pages 157–168. ACM, 2019.
75. Xiaowu Sun, Haitham Khedr, and Yasser Shoukry. Formal verification of neural network controlled autonomous systems. In *International Conference on Hybrid Systems: Computation and Control*, pages 147–156. ACM, 2019.
76. Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. Reachability analysis of deep neural networks with provable guarantees. In *IJCAI*, pages 2651–2659, 2018.
77. Wenjie Ruan, Min Wu, Youcheng Sun, Xiaowei Huang, Daniel Kroening, and Marta Kwiatkowska. Global robustness evaluation of deep neural networks with provable guarantees for L0 norm. *arXiv preprint arXiv:1804.05805*, 2018.
78. M. Wu, M. Wicker, W. Ruan, X. Huang, and M. Kwiatkowska. A game-based approximate verification of deep neural networks with provable guarantees. *arXiv preprint arXiv:1807.03571*, July 2018.

Appendix

This section gives the proofs for relations given in Section 4.

Lemma 2. $M_N \preceq M_{SS}$.

Proof. Note that, for every hidden node $n_{k,j} \in \mathcal{H}(\mathcal{N})$, there exists a feature pair $(\{n_{k-1,i}\}, \{n_{k,j}\}) \in \mathcal{O}(\mathcal{N})$ for any $1 \leq i \leq s_{k-1}$. Then, by Definition 4, we have $sc(\{n_{k,j}\}, x_1, x_2)$, which by Definition 2 means that $sign(n_{k,j}, x_1) \neq sign(n_{k,j}, x_2)$. That is, either $sign(n_{k,j}, x_1) = +1$ or $sign(n_{k,j}, x_2) = +1$. Therefore, if $n_{k,j}$ is not covered in a test suite $\mathcal{T}_1$ for neuron coverage, none of the pairs $(\{n_{k-1,i}\}, \{n_{k,j}\})$ for $1 \leq i \leq s_{k-1}$ is covered in a test suite $\mathcal{T}_2$ for SS coverage.

Lemma 3. $M_N \preceq M_{VS^g}$.

Proof. Follow a similar argument with Lemma 2.

Lemma 4. $M_N \preceq M_{MN^m}$*, when the interval* $[v^l_{k,i}, v^u_{k,i}]$ *is non-trivial, i.e., not* $[0, 0]$*, for all nodes* $n_{k,i} \in \mathcal{H}$*.*

Proof. Because m subsections are all in $[v^l_{k,i}, v^u_{k,i}]$ and $[v^l_{k,i}, v^u_{k,i}]$ is non-trivial, we have that the neuron coverage of a node $n_{k,i}$ is satisfied whenever any subsection of $[v^l_{k,i}, v^u_{k,i}]$ is filled.

We remark that the condition about the non-trivial intervals are reasonable. First of all, in practice, all the DNNs we work with satisfy this condition. Second, if a node always have value 0 for all the training samples then such a node can be seen as redundant.

Lemma 5. $M_{MN^m} \preceq M_{SV^g}$ *for a suitable function g.*

Proof. For every hidden node, the upper bound $v^u_{k,i}$ and lower bounds $v^l_{k,i}$ are obtained from the training samples. Therefore, for any given subsection of $[v^l_{k,i}, v^u_{k,i}]$, we know its exact interval, say $[v^{l,j}_{k,i}, v^{u,j}_{k,i}]$ for the j-th subsection. Then we can use the function g to express that the value $v_{k,i}[x_2]$ is in $[v^{l,j}_{k,i}, v^{u,j}_{k,i}]$. The value of $v_{k,i}[x_1]$ does not matter. Therefore, if the subsections $[v^{l,j}_{k,i}, v^{u,j}_{k,i}]$ is not covered, we know that the feature pairs $(\{n_{k-1,j}\}, \{n_{k,i}\})$ for $1 \leq j \leq s_{k-1}$ have not been covered by the SV coverage under the function g.

Lemma 6. $M_{MN^m} \preceq M_{VV^{g_1,g_2}}$ *for a suitable function g_2.*

Proof. Follow a similar argument with Lemma 5.

Lemma 7. $M_{NB} \preceq M_{SV^g}$ *for a suitable function g.*

Proof. Follow a similar argument with Lemma 5, except that in this case the function g is used to express that $v_{k,i}$ is greater than $v^u_{k,i}$.

Lemma 8. $M_{NB} \preceq M_{VV^{g_1,g_2}}$ *for a suitable function g_2.*

Proof. Follow a similar argument as that of Lemma 7.

Lemma 9. $M_{TN^m} \preceq M_{VV^{g_1,g_2}}$ *for a suitable function* g_1.

Proof. We can work with feature pairs $(\{n_{k,i}, n_{k+1,j}\})$ and use g_1 to express that $v_{k,i}[x_2]$ is greater than at least $s_k - m$ values in the set $\{v_{k,l} \mid l \in [1..s_k]\}$. The value of $v_{k,i}[x_1]$ does not matter. Therefore, whenever the node $n_{k,i}$ is not top-m neuron covered then the pair is not VV covered under the functions g_1 and g_2 for any g_2.

Lemma 10. $M_{TN^m} \preceq M_{VS^g}$ *for a suitable function* g_1.

Proof. Follow the similar argument with that of Lemma 9.

We have the following conclusion stating the relationship between safety coverage and ours.

Theorem 1. $M_{SS} \preceq M_S$.

Proof. Note that, safety coverage exhaustively enumerates all possible activation patterns. Therefore, we have $M_{SS} \preceq M_S$ since the former only explore a subset of the activation patterns.