Structural Test Coverage Criteria for Deep Neural Networks

YOUCHENG SUN, Queen's University Belfast
XIAOWEI HUANG, University of Liverpool
DANIEL KROENING, University of Oxford
JAMES SHARP, MATTHEW HILL, and ROB ASHMORE, Defence Science and Technology Laboratory (Dstl)

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 coverage criteria that are tailored to structural features of DNNs and their semantics. We validate the criteria by demonstrating that test inputs that are generated with guidance by 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 correlation with functional coverage) and the computational cost of test input generation. Our experiments are conducted on state-of-the-art DNNs obtained using popular open source datasets, including MNIST, CIFAR-10 and ImageNet.

CCS Concepts: • **Software and its engineering → Software testing and debugging**;

Additional Key Words and Phrases: Neural networks, test criteria, test case generation

ACM Reference format:
Youcheng Sun, Xiaowei Huang, Daniel Kroening, James Sharp, Matthew Hill, and Rob Ashmore. 2019. Structural Test Coverage Criteria for Deep Neural Networks. *ACM Trans. Embed. Comput. Syst.* 18, 5s, Article 94 (October 2019), 23 pages.
https://doi.org/10.1145/3358233

1 INTRODUCTION

Artificial intelligence (AI), and 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.

This article appears as part of the ESWEEK-TECS special issue and was presented at the International Conference on Embedded Software (EMSOFT) 2019.
Authors' addresses: Y. Sun, Queen's University Belfast, 18 Malone Road, Belfast, BT9 6RT; email: youcheng.sun@qub.ac.uk; X. Huang, University of Liverpool, Ashton Street, Liverpool, L69 3BX; email: xiaowei.huang@liverpool.ac.uk; D. Kroening, University of Oxford, Wolfson Building, Parks Road, Oxford, OX1 3QD; email: kroening@cs.ox.ac.uk; J. Sharp, M. Hill, and R. Ashmore, Defence Science and Technology Laboratory (Dstl); emails: {jsharp1, mhill2, rdashmore}@dstl.gov.uk.

1539-9087/2019/10-ART94 $15.00
https://doi.org/10.1145/3358233

Research and industrial communities worldwide are making significant efforts towards the best practice for safety assurance for learning-enabled autonomous systems. Among all efforts, we mention a proposal under consideration by IEEE to form an official technical committee for verification of autonomous systems [1]. Moreover, as stated in [29], the learnt algorithm should be verified with an appropriate level of coverage. This paper develops a technical solution to support these efforts.

The software industry relies on testing as a primary means to provide stakeholders with information about the quality of a software product or service [17]. Research in software testing has resulted in a broad range of approaches to assess software at different criticality levels. In white-box testing, the structure of a program is exploited to (perhaps automatically) create test cases. Code coverage criteria (or metrics) have been designed to quantify 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 to what extent coverage ensures correct functionality, high coverage is able to increase users' confidence (or trust) in the program. Structural coverage metrics are used as a means of assessment in several high-tier safety standards; for instance, DO-178C requires MC/DC coverage for function bodies [28]. MC/DC was developed by NASA, and is used in avionics software development guidance to ensure adequate testing of applications with the highest criticality.

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 typically simplistic and is unable to capture the knowledge that is learned during the training phase. The definition of useful structural coverage criteria for DNNs is therefore non-trivial [4]. Meanwhile, DNNs exhibit "bugs" that differ to those in traditional software. Notably, *adversarial examples* [34], in which two apparently indistinguishable inputs yield contradicting decisions, are a prominent safety concern in DNNs.

We believe that the testing of DNNs, guided by proper coverage criteria, must help developers to find those bugs; it has to be able to quantify the robustness of the network and it needs to support the analysis of the internal structures of the DNN. The tests should enable developers to understand and compare different networks and should be able to support safety-related arguments.

Technically, DNNs feature 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 activation functions used and the semantic relationship between layers.

In this paper we propose a novel white-box testing methodology for feedforward DNNs. In particular, we propose a family of four test coverage criteria, inspired by the MC/DC test coverage criterion [14] 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 [27] can be achieved by a simple test suite comprised of a few input vectors from the training dataset. Conversely, an overly strong criterion may lead to computational intractability; e.g., 100% safety coverage is shown to be difficult to achieve in [39]. Our criteria, when applied to guide test case generation, deliver both appropriate testing (i.e., it is non-trivial to achieve 100% coverage) and are computationally feasible. As a matter of fact, except for the safety coverage criterion in [39], all existing structural test coverage criteria for DNNs [21, 27] are special cases of our proposed criteria. Our criteria are the first that capture and quantify the causal relationships in a DNN that are critical for understanding the behaviour of the neural network [25, 41].

Subsequently, we validate the utility of our MC/DC variant by applying it to different approaches to DNN testing. First, we implement state-of-the-art concolic testing for DNNs [33]. Concolic testing combines concrete testing with a symbolic encoding. Specifically, a linear programming (LP) based algorithm produces a new test case (i.e., an input vector) by encoding a fragment of the DNN and then minimises 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. The LP test generation algorithm does not apply to DNNs with tanh or sigmoid activation functions, or to DNNs that are very large. We have therefore developed a gradient descent based algorithm that takes the test condition as the optimisation objective, and searches for test cases in an adaptive manner under the guidance of the first-order derivative of the DNNs; this method is able to scale to large DNNs and is a good fit for tanh or sigmoid activation functions.

Finally, we experiment with our test coverage criteria on state-of-the-art neural networks that have a broad range of sizes (from a few hundred up to millions of neurons) to demonstrate the utility of our criteria with respect to four aspects: bug finding (proxied by adversarial examples), their ability to quantify the safety of a DNN, the efficiency of testing input generation, and whether they can support the analysis of the structure of a DNN.

2 PRELIMINARIES: DEEP NEURAL NETWORKS

A (feedforward and deep) neural network [12], or DNN, is a tuple $\mathcal{N} = (L, T, \Phi)$, where $L = \{L_k \mid k \in \{1, \ldots, K\}\}$ is a set of layers, $T \subseteq L \times L$ is a set of connections between the layers and $\Phi = \{\phi_k \mid k \in \{2, \ldots, K\}\}$ is a set of functions, one for each non-input layer. In a DNN, L_1 is the *input* layer and L_K is the *output* layer; the other 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 the values before and after an activation function, respectively. ReLU [23] is by far the most popular activation function for DNNs, according to which the *activation value* of each node of hidden layers is

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

Each input node $n_{1,l}$ for $1 \leq l \leq s_1$ is associated only with a variable $v_{1,l}$ and each output node $n_{K,l}$ for $1 \leq l \leq s_K$ is associated only 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 $v_{k,l}$. Every point $x \in D_{L_1}$ is a possible input.

Except for the input nodes, every node is connected to nodes in the preceding layer by 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 given in (1), the behavior of the neural network is highly non-linear.

Finally, for any input, the DNN assigns a *label*, which is the index of the node of the output layer with the largest value: *label* $= \text{argmax}_{1 \leq l \leq s_K} u_{K,l}$. 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.

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

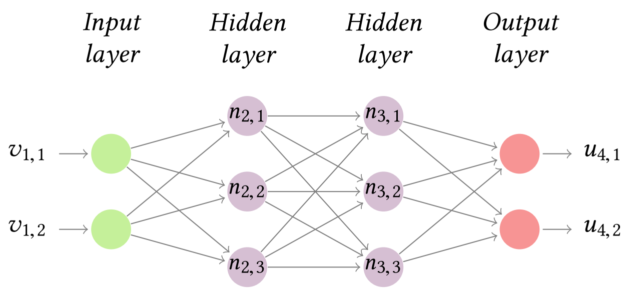


Fig. 1. A simple neural network.

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 in Figure 1. Assume that the weights for the first two layers are $W_1 = \begin{bmatrix} 4 & 0 & -1 \\ 1 & -2 & 1 \end{bmatrix}$ 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 $u_{2,2}[x] = -2 \neq 0 = v_{2,2}[x]$.

To keep the discussion simple, the presentation 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 max-pooling layers, and sigmoid activation functions used in state-of-the-art DNNs, which are not fully connected.

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 using a set of test coverage conditions. Throughout this paper, we use "criterion" and "metric" interchangeably.

Our coverage criteria for DNNs are inspired by established practices in software testing, and in particular the MC/DC test coverage criterion [14], but are designed for the specific attributes of DNNs. MC/DC is a method for measuring the extent to which safety-critical software has been 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

$$((a > 3) \lor (b = 0)) \land (c \neq 4) \tag{4}$$

contains the conditions $(a > 3)$, $(b = 0)$ *and* $(c \neq 4)$. *The following four test cases provide full 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 provide both condition coverage *(i.e., all possibilities for the conditions are exercised) and* decision coverage *(i.e., both outcomes for the decision are exercised). The last two test cases are needed for MC/DC because 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 semantics 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. It has been claimed that nodes in a deeper layer (having the larger layer index) represent "more complex" attributes of the input [25, 41].

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. That is, Ψ_k is the set of features and any $\psi_{k,l} \in \Psi_k$ is a feature. 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 $1 \leq l \leq t_k$ to denote the set of nodes of the l-th feature. Features can be overlapping, i.e., there may be l_1 and l_2 with $\psi_{k,l_1} \cap \psi_{k,l_2} \neq \emptyset$. 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., $\{\psi_{k-1,l'} \mid 1 \leq l' \leq t_{k-1}\}$. For simplicity, the DNN model described in Section 2 only considers a fully connected structure. In the more general case, two features in adjacent layers of a DNN need not be connected, and thus there is no causal effect between such feature pairs.

The concept of a "feature" generalises the basic building block in the DNN from a single node to a set of nodes. A single node feature can be represented as a singleton set. In practice, this definition of "feature" is a good fit for the tensor implementation in popular machine learning libraries [2] and there are a variety of applicable feature extraction methods such as SIFT [20], SURF [5], 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 the activation function, 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* if both $\psi_{k,i}$ and $\psi_{k+1,j}$ are singleton sets.

Our new criteria are defined by instantiating the definitions of what it means to "change" the result of a condition and of a decision in different ways. Unlike Boolean variables or expressions, where it is obvious what a "change" is, i.e., true becomes false or false becomes 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 triggered 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 $nsc(\psi_{k,l}, x_1, x_2) \neq \neg sc(\psi_{k,l}, x_1, x_2)$. That is, *nsc* is one special case when *sc* does not hold.

Before proceeding to another kind of change called *value change*, we need notation for the value function. A value function is denoted by $g : \Psi_k \times D_{L_1} \times D_{L_1} \rightarrow \{\text{true}, \text{false}\}$. Simply speaking, it expresses the intuition (or knowledge) of the developer of the DNN 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 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 be a constraint on one of the values $u_{k,j}[x_2]$, say an upper bound $u_{k,j}[x_2] > d$.

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 [38]. It can also express constraints between nodes of the same layer, such as $\bigwedge_{j \neq i} v_{k,i}[x_1] \geq v_{k,j}[x_1]$.

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, the appropriate choice is likely domain specific. Finally, we define "value change" as follows.

Definition 3 (Value Change). Given a feature $\psi_{k,l}$, two inputs x_1 and x_2, and a value function g, the value change of $\psi_{k,l}$ w.r.t. g is triggered by x_1 and x_2, denoted by $vc(g, \psi_{k,l}, x_1, x_2)$, if $g(\psi_{k,l}, x_1, x_2)$=true. We write $\neg vc(g, \psi_{k,l}, x_1, x_2)$ when this condition is not satisfied.

3.3 Coverage Criteria

In this section, we present a family of four criteria that capture the state 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 triggered by 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 triggered by 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

Table 1. Activation Values and Sign Changes for the Inputs in Examples 6, 7, 8, 9

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

An entry has 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.

has additional complexity that is not necessarily captured by a direct adoption of the MC/DC-style coverage to a DNN. We now give three additional coverage criteria to complement SS coverage.

First, the sign of $\psi_{k+1,j}$ may change when transitioning from one test input to another, 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 $nsc(P_k, x_1, x_2)$;
- $sc(\psi_{k+1,j}, x_1, x_2)$.

Intuitively, the first condition describes the value change of nodes in layer k and the second requires 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 $nsc(P_k, x_1, x_2)$, which prevents sign changes for any node at layer k. This is to ensure that the overall change to the activation pattern in layer k is small.

Example 7 (Continuing 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 no node in layer 2 changes its activation sign, 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 (given a 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 treated 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 [34] 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 in the output (e.g., its label). We expect that our criteria can guide test case generation algorithms towards such 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 the sign of one of its condition features. Intuitively, SV coverage captures the significant change of a decision feature's value and complements the sign change case.

Example 8 (Continuing 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., $g(u_{3,2}[x_1], u_{3,2}\ [x_2]) = \text{true}$, then the pair $(\{n_{2,1}\}, \{n_{3,2}\})$ is SV-covered by x_1 and x_2.

Finally, we obtain 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 in which the value of a decision feature changes significantly.

Example 9 (Continuation of Example 2). For any $i \in \{1, \ldots, 3\}$, the feature pair $(\{\psi_{2,i}\}, \{\psi_{3,3}\})$ is 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 $i \in \{1, \ldots, 3\} : nsc(\{n_{2,i}\}, x_1, x_2)$ and $nsc(\{n_{3,3}\}, x_1, x_2)$.

We present four representative variants that best express our idea in terms of the features and the relationship between the features in neighbouring layers. While we agree that this choice is empirical and that there exist other ways to identify causal changes in a neural network, our particular choice is based on the principle that a) the operations between two adjacent DNN layers drive the choice of features and b) that there is a straightforward relationship between the features in two adjacent layers.

3.4 Coverage Metrics

Using the definitions in Section 3.3, we now are able to define the set of properties that a suite of test cases needs to satisfy for each of our coverage criteria. Let $F = \{SS, VS^g, SV^g, VV^{g_1,g_2}\}$ be our set of criteria, and let $f \in F$ be one of them. A test suite comprises of a sequence of test inputs and expected test outputs; our coverage criteria do not consider the expected outputs, and thus, for the purpose of defining the metric for the case of a stateless network we can view a test suite $\mathcal{T}$ as a finite and unordered set of test inputs, i.e., $\mathcal{T} \subseteq D_{L_1}$. Let $\mathcal{O}$ denote the set of pairs of neurons in a DNN. A test suite $\mathcal{T}$ for that DNN satisfies criterion f completely if

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

In practice, coverage for all feature pairs is rarely achieved; we therefore compute the degree to which the test condition f is satisfied by the test suite $\mathcal{T}$.

Definition 8 (Coverage Metric). Given a DNN $\mathcal{N}$, a test condition fixed by $(f, \mathcal{O})$ and a test suite $\mathcal{T}$, the value of the coverage metric $M_f(\mathcal{N}, \mathcal{T})$ is

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

That is, we compute the percentage of the feature pairs that are covered by test cases in $\mathcal{T}$ with respect to the coverage criterion f. Finally, instantiating f with one of the criteria in F, we obtain four coverage metrics $M_{SS}(\mathcal{N}, \mathcal{T})$, $M_{VS^g}(\mathcal{N}, \mathcal{T})$, $M_{SV^g}(\mathcal{N}, \mathcal{T})$ and $M_{VV^{g_1,g_2}}(\mathcal{N}, \mathcal{T})$.

4 COMPARISON WITH EXISTING STRUCTURAL TEST COVERAGE CRITERIA

There have already been proposals for structural test coverage criteria for DNNs. In this part, we compare our new criteria with safety coverage (M_S) [39], neuron coverage (M_N) [27] and several of its extensions given in [21], including neuron boundary coverage (M_{NB}), multisection neuron coverage (M_{MN}) and top neuron coverage (M_{TN}). While [27] and [39] have been published slightly ahead of our work, our criteria have been developed in parallel with [21].

A metric M_1 is said to be "weaker than" (or is "subsumed by") 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 strictly weaker than MC/DC, since MC/DC coverage cannot be achieved before all decisions and conditions are covered.

The introduction of the feature relation in this work is very powerful, for two reasons: 1) the criteria in this paper are stronger than those in [27] and [21], which only consider the activation status of individual neurons, and 2) it is non-trivial for safety coverage [39], which is comparable to traditional path coverage that requires covering every program execution path, to satisfy all test conditions of our criteria.

In the following, we give a uniform formalisation of the criteria from [21, 27, 39], using the notation introduced in this paper, and we 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 input x, denoted by $N(n_{k,i}, x)$, if $sign(n_{k,i}, x) = +1$.

Informally, neuron coverage requires that each neuron $n_{k,i}$ must be activated at least once by some test input. Neuron coverage was later generalised in [21] using more fine-grained neuron activation conditions, 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 input 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., it refers to the relative activation magnitude of the neuron under input x among all values in $\{v_{k,j}[x] \mid 1 \leq j \leq s_k\}$.

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 j-th 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 $\mathcal{N}$ (the nodes of hidden layers of $\mathcal{N}$), their associated coverage metric 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 show that the criteria in [27, 39] are special cases of our criteria (with a suitable value function g). As exemplar, the "weaker than" relationship between neuron coverage and SS coverage is proved in the lemma below.

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

Proof. 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 by a test suite $\mathcal{T}_2$ that provides SS coverage. □

In [39], the input space is discretised with a set of hyper-rectangles, and then one test case is generated for each hyper-rectangle. This is referred to as "safety coverage".

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

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

Such a scheme is computationally intractable because of the high dimensionality of DNNs. The approach to testing in this paper aims to be more practical.

Figure 2 gives a diagrammatic summary of the relations between all existing structural test coverage criteria for DNNs. The arrows represent the "weaker than" relation. The complete proofs are in a longer version of this paper [32]. As indicated in Figure 2, our criteria require more test cases to be generated than those in [21, 27], and therefore can make testing more exhaustive. On the other hand, SS coverage is weaker than safety coverage [39].

5 AUTOMATED COVERAGE-DRIVEN 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 to drive two different test case generation approaches for DNNs.

The test conditions required by our criteria exhibit particular combinations of the condition feature and the decision feature, and it is not trivial to generate test cases for them. Owing to the lack of awareness of the feature relation, the test input generation methods in [21, 27, 39] cannot be directly used to generate tests that satisfy our criteria. Also, as pointed out in [24], random test case generation is prohibitively inefficient for DNNs. Conversely, the symbolic encoding that is used in the concolic testing method in [33] is expressive enough to encode the test conditions defined by our criteria and is suitable for small- to medium-sized DNNs. Furthermore, in this section, we

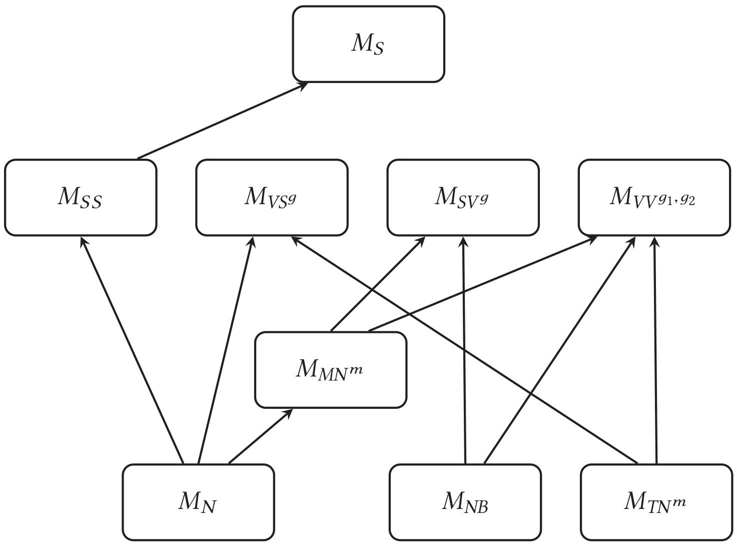


Fig. 2. Subsumption relationship between several structural test coverage criteria for DNNs.

also present a new test case generation algorithm based on gradient descent 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 $\mathcal{F}(x) : D_{L_1} \rightarrow \mathcal{L}$, which models perfect perception. 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, it is not practical to use this requirement to define the oracle 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 [15, 34], is to use the following oracle as an inexpensive proxy.

Definition 14 (Oracle). Given a finite set X of correctly labeled inputs, a test with input x' passes 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 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. An *adversarial example* is detected if there is a pair of inputs satisfying this definition such that the two labels assigned to them by the DNN differ and either x or x' is in the training or validation dataset.

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 5 for one of the criteria). We remark that there may exist other ways to define a test oracle for DNNs, and that our coverage criteria are independent from its particular definition.

5.2 Test Case Generation with LP

We first apply the concolic testing approach in [33] to generate test inputs that satisfy the test conditions defined by our criteria. In [33], the test conditions are symbolically encoded using a linear programming (LP) model, which 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.

While the overall behaviour of a DNN is highly non-linear, owing to the use of the ReLU activation function, once the DNN is instantiated with a particular input, the activation pattern is fixed, and we obtain an LP model.

LP model of a DNN instance. The variables used in the LP model are typeset in **bold**. All variables range over the rational numbers. Given an input x, the input variable $\mathbf{x}$, whose value is to be synthesized with LP, is required to exhibit the same 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 $C_1[x]$ of constraints to encode ReLU operations for a network instance, where $C_1[x]$ 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 that the activation values $\mathbf{u}_{k,i}$ of the nodes are determined by the activation values $\mathbf{v}_{k-1,j}$ of the nodes in the previous layer, following Equation (2). Therefore, we add the following set of constraints, $C_2[x]$, as a symbolic encoding of the activation values of the nodes.

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

The resulting LP model $C[x] = C_1[x] \cup C_2[x]$ represents *a symbolic set of inputs* that have the same 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 Heuristic Test Input Generation Using Gradient Search

The LP optimisation method in Section 5.2 provides a guarantee that an input pair is returned as long as one exists. However, its scalability depends on the efficiency of the LP solvers, and it is not trivial to apply this method to DNNs with millions of neurons. In this part, we offer a heuristic algorithm based on gradient search as an alternative with better scalability. It has been shown that gradient search is an efficient method for finding adversarial examples in DNNs. This approach has been utilised in a number of existing DNN testing methods (e.g., [21, 27, 40]).

ALGORITHM 1: *get_input_pair*$(f, \psi_{k,i}, \psi_{k+1,j})$

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

The procedure, given as Algorithm 1, finds an input pair 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. In principle, there are two objectives in the search algorithm: requirements on the feature pair $(\psi_{k,i}, \psi_{k+1,j})$, and requirements on other feature pairs. For example, in the case of SS coverage, this means *sc* for $\psi_{k,i}$ and $\psi_{k,2}$ and *nsc* for all other features. To simplify the search, we use $f^{widen}(\alpha, x_1, x_2)$ as a relaxed version of the testing condition f, such that all its predicates on the features $\psi_{k,i}$ and $\psi_{k+1,j}$ are eliminated. Given some original input x_1, and starting from an input x_2, if feature changes other than $\psi_{k,i}$ and $\psi_{k+1,j}$ are too prevalent, and do not meet the requirements of f^{widen} (Line 6), then x_2 is moved closer to x_1, by following the gradient descent $x_2 \leftarrow x_2 - \epsilon \cdot \nabla\hat{\mathcal{F}}(x_2)$, in an attempt to counteract such feature 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 trigger a subset of the predicates (in the testing condition) from the given feature pair, and we thus need to update x_2 following the gradient ascent (Line 7). The algorithm's gradient change follows an adaptive manner that comprises: 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) [13] to initialise x_2 and ϵ, and use a binary search scheme to update ϵ at each step.

As a heuristic, the algorithm works well when there exist 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, the CIFAR-10 dataset with low-resolution color images and the ImageNet benchmark from the large-scale visual recognition challenge. Our experiments are classified into four categories: ① *bug finding,* ② *quantification of the safety of a DNN,* ③ *efficiency of testing, and* ④ *the analysis of the internal structure of DNNs.* We also explain the relationship between our criteria and the existing ones. All implementations and the data discussed in this section are available online[1].

In our implementation, the objective $\min ||x_2 - x_1||_\infty$ is used in all LPs 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 empirical. 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.

6.1 MNIST

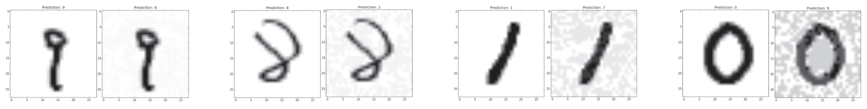
We randomly generate, and then train, a set of ten fully connected DNNs, such that each network has an accuracy of at least 97.0% on the MNIST validation data. The detailed network structure, and the number of neurons per layer, are given in Table 3. Every DNN input has been normalised into $[0, 1]^{28\times28}$. Experiments were conducted on a MacBook Pro (2.5 GHz Intel Core i5 and 8 GB of memory).

We apply the coverage criteria defined in Section 3. Besides the coverage M_f, we also compute the percentage of adversarial examples in the test suite, denoted by AE_f. *The use of LP optimisation enables us to use single neurons as features.* That is, each feature pair is in fact a neuron pair. The choice of pairs of neurons is the most difficult scenario for test input generation.

[1] https://github.com/TrustAI/DeepConcolic.

	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%

Fig. 3. Coverage results on ten DNNs.



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

Fig. 4. Selected adversarial examples for MNIST.

DNN Bug finding ①. The results, reported in Table 3, are promising: (1) the test case generation algorithm achieves high coverage effectively for all coverage criteria, and (2) the coverage criteria are useful, indicated by the fact that a significant number of adversarial examples are identified among the generated test cases. Figure 4 exhibits several adversarial examples found by the tool with different distances. We note that, *for neuron coverage [27], high coverage can be achieved trivially by selecting a few non-adversarial test cases that we generated.*

Quantifying the safety of a DNN ②. The coverage M_f and adversarial example percentage AE_f together provide quantitative means to evaluate the safety of 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 examples are to the corresponding original input. The quality of an adversarial example is measured by its distance to the original input. An adversarial example is of higher quality than others if its distance to the original input is smaller. Consider the results for SS coverage for the last three DNNs in Table 3. In the graph given as Figure 5, the horizontal axis measures the L^∞ distance and the vertical axis gives the accumulated percentage of the adversarial examples within this distance. A more robust DNN will exhibit a curve where most adversarial examples have a small distance. Intuitively, this means that more effort needs to be made to fool such a DNN.

Understanding the behavior of individual layers ④. Our experiments suggest that different layers of a DNN exhibit different behaviors during testing. Figure 6 reports the SS coverage results, collected in adjacent layers. In particular, Figure 6(a) gives the percentage of covered neuron pairs within individual adjacent layers. We observe that, when going deeper into the DNN, it can become harder to cover neuron pairs. Under such circumstances, to improve coverage, a larger data set is needed when generating pairs of test inputs. Figure 6(b) gives the percentage of adversarial examples found at different layers (among all adversarial examples detected). Interestingly, it seems that most adversarial examples are found when testing the middle layers.

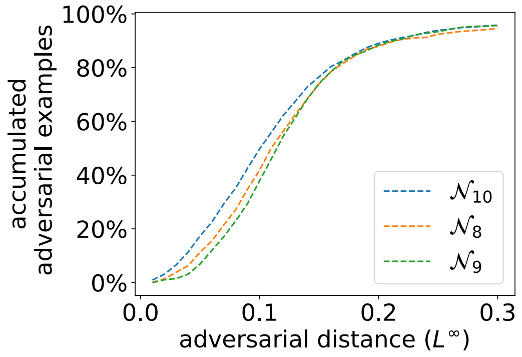


Fig. 5. Accumulated percentage of adversarial examples (y-axis) that do not exceed a particular distance (given on the x-axis): the adversarial distance measures the distance between an adversarial example and the original input.

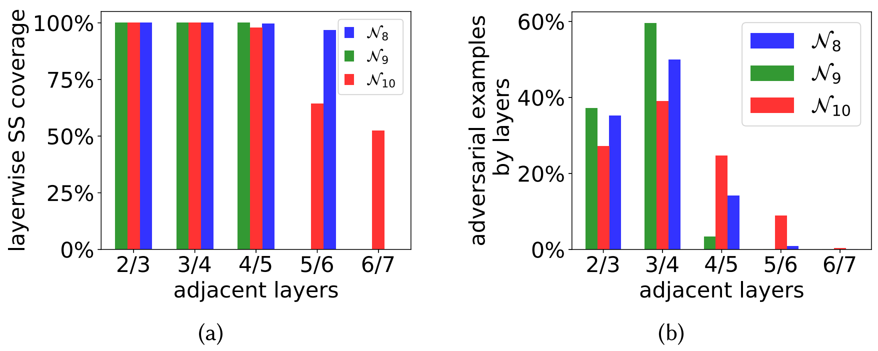


Fig. 6. SS coverage by layer: (a) the coverage level per DNN layer; (b) the detected adversarial examples at each layer with respect to the total amount of adversarial examples.

SS coverage with top weights ③. For SS coverage criteria with neuron pairs, there are $|O|$ test conditions for $O \subseteq \mathcal{O}(\mathcal{N})$ in total. 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 $\{|w_{k,i',j}| \mid i' \in [1 \ldots s_k]\}$. 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 7 shows the difference of the coverage and adversarial example percentages when comparing 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 reduces the size of the resulting test suite; thus, the simplification can be used as a fast pre-processing phase and can even deliver performance that is comparable to that of SS coverage.

Cost of LP calls ③. Since the LP encoding of the (partial) activation pattern plays a key role in the test generation, we give details of the cost of the LP calls. For every DNN, we select a set of neuron pairs, where each decision neuron is in 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. We use CPLEX as the LP solver. The results in Table 2 confirm that the LP model of a partial activation pattern is indeed lightweight, and that its complexity increases in a linear manner when traversing into deeper layers of a DNN.

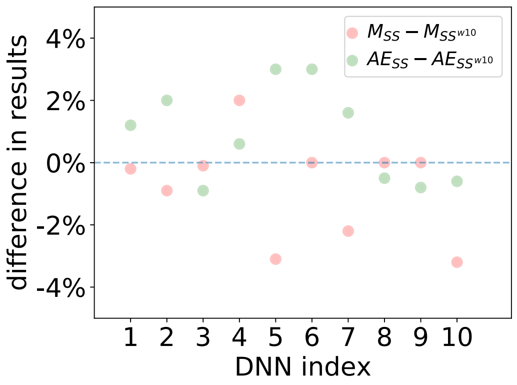


Fig. 7. *SS* vs. SS^{w10}. Results demonstrate that the SS coverage and its top-weight simplification have similar coverage levels ($M_{SS} - M_{SS^{w10}}$) and percentages of adversarial examples ($AE_{SS} - AE_{SS^{w10}}$).

Table 2. Number of Variables and Constraints, and Time Cost of Each LP Call (in seconds)

	$\mathcal{N}_8$			$\mathcal{N}_9$			$\mathcal{N}_{10}$		
	#vars	$\lvert C\rvert$	t	#vars	$\lvert C\rvert$	t	#vars	$\lvert C\rvert$	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

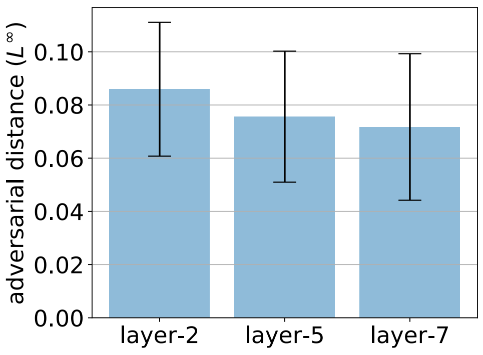


Fig. 8. Average adversarial distance for decision features at different layers.

6.2 CIFAR-10

The CIFAR-10 dataset is a collection of 32×32 color images with ten kinds of objects. In contrast to the MNIST case, we need to train a DNN with convolutional layers in order to handle the CIFAR-10 image classification problem. We apply the test case generation in Algorithm 1 for the SS coverage and measure the coverage results individually for the decision features at each different layer. Overall, an SS coverage higher than 90% is achieved with a substantial average 84.36% of the tests generated being adversarial examples. An interesting observation can be made in Figure 8 (④), which shows that in this case the causal changes of features at deeper layers are able to detect

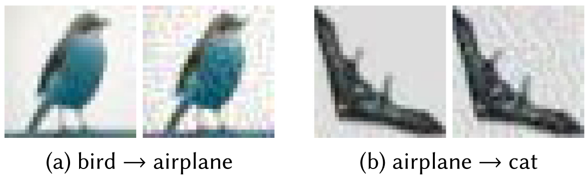


Fig. 9. Two selected adversarial examples for CIFAR-10.

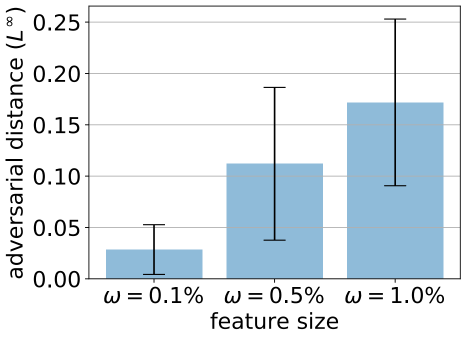


Fig. 10. Adversarial distance with different feature sizes: a smaller distance corresponds to more subtle adversarial examples.

smaller perturbations of inputs that cause adversarial behaviours. This could be helpful feedback for developers for debugging or tuning the parameters of the neural network. Selected adversarial examples are given in Figure 9.

6.3 ImageNet

We applied our methods to VGG16 [30], a large-scale DNN trained on the ImageNet dataset. The heuristic search Algorithm 1 is called to generate test cases. We consider every neuron a decision feature. While feature extraction methods such as SIFT [20] can obtain condition features, in our experiments we use arbitrary sets of neurons as conditions. We define a size parameter ω and require feature $\psi_{k,i}$ to have size $\leq \omega \cdot s_k$. Recall that s_k is the number of neurons in layer k.

Effect of feature size ① ② ④. We apply SS coverage on 2,000 randomly sampled feature pairs with $\omega \in \{0.1\%, 0.5\%, 1.0\%\}$. The test case generation method shows its effectiveness by returning a test suite in which 10.5%, 13.6% and 14.6% of the tests are adversarial examples, respectively. We report the average distance of the adversarial examples and the standard deviation in Figure 10. The results confirm that there is a relationship 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.

The results in Figure 10 are obtained using the L^∞-norm, which corresponds to the maximum change to a pixel. We observe that, although the change of each pixel is very small, for every adversarial example a large portion (around 50%) of the pixels are changed. A typical adversarial example image is given in Figure 11. Overall, the detected adversarial examples are high quality.

SV with neuron boundary coverage ① ②. As shown in Section 4, our coverage criteria are strictly 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 value exceed the specified boundary value. We confirm this claim empirically, similarly to the experiments above,

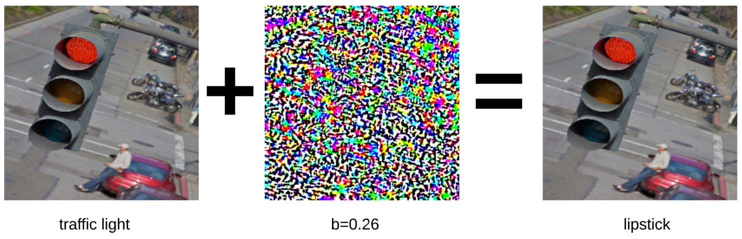


Fig. 11. An adversarial example ("lipstick") for the original traffic light input.

by generating a test suite using SV with neuron boundary coverage. We noticed that reaching boundary activation values requires substantial changes to the inputs. 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 illustrated in Figure 10.

6.4 Comparing DNNs Using Testing

We discuss whether testing can inform a choice between alternative DNN models. In our experiment, we choose two DNNs such that one is apparently better than the other, and then apply coverage-guided testing and compare the results.

Complex vs. simple features. The recognition of features is a fundamental building block in deep learning, and the convolutional kernel operation in a neural network is the most basic approach to define a feature. Without loss of generality, a convolutional DNN can be treated as a neural network such that the activation of a node in layer k is computed by the activations of a subset of precedent nodes, and each node belongs to a feature map in its layer. Hence, each node $n_{k,i}$ is a feature that abstracts the preceding features such that each preceding feature at layer $(k-1)$ is a subset of nodes from the same feature map that are connected to $n_{k,i}$. Nodes in the same feature map share the same weights and bias that is given by a convolutional *filter* of some shape $d_1 \times d_2$ [12].

We have trained two convolutional DNNs $\mathcal{N}_1^c$ and $\mathcal{N}_2^c$. Both have two convolutional layers followed by one fully connected layer of 128 neurons. $\mathcal{N}_1^c$ (resp. $\mathcal{N}_2^c$) has 20 (resp. 2) filters for the first convolutional layer and 40 (resp. 4) filters for the second convolutional layer. The filters have size 5×5 and every convolutional layer is augmented with a so-called max-pooling layer of size 2×2. Overall, $\mathcal{N}_1^c$ has 14,208 hidden neurons, and $\mathcal{N}_2^c$ is much smaller with only 1,536 hidden neurons. We say that $\mathcal{N}_1^c$ is "complex" and that $\mathcal{N}_2^c$ is "simple".

In the experiment, we use SS coverage to guide the testing of neuron pairs from feature maps at adjacent layers. We use $f_{k,i}$ to denote the i-th feature map in the k-th convolutional layer, and $M_{SS^{f_{k,i}}_{f_{k+1,j}}}$ represents the SS coverage among neuron pairs such that the condition neuron and decision neuron are from $f_{k,i}$ and $f_{k+1,j}$, respectively. Correspondingly, $AE_{SS^{f_{k,i}}_{f_{i+1,j}}}$ denotes the percentage of adversarial examples for each feature map. The results are in Table 3. We obtain high SS coverage for both architectures. The generated test suite confirms that the use of the complex convolutional structures is justified, as the simple architecture is much more susceptible to adversarial examples.

To cross-validate this result, we conduct an experiment using the adversarial attack algorithm FGSM (Fast Gradient Sign Method) [13]. We apply FGSM to distort the default MNIST validation data set. Figure 12 gives the number of adversarial examples in the new data set for both neural

Table 3. SS Coverage Results per Feature Map

	$M_{SS}^{(1,1),(2,1)}$	$AE_{SS}^{(1,1),(2,1)}$	$M_{SS}^{(1,1),(2,2)}$	$AE_{SS}^{(1,1),(2,2)}$
$\mathcal{N}_1^c$	99.7%	1.6%	99.7%	1.5%
$\mathcal{N}_2^c$	100.0%	7.6%	100.0%	6.8%

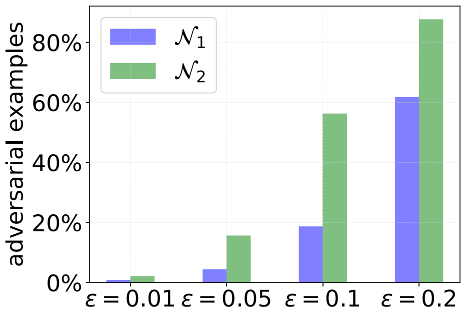


Fig. 12. Adversarial examples found by FGSM.

networks $\mathcal{N}_1^c$ and $\mathcal{N}_2^c$, based on the distorted data; ϵ is the parameter that controls the degree of change of the original inputs. The results in Figure 12 are consistent with ours.

A large number of methods based on gradient search have been proposed; these methods motivate our test generation algorithm in Section 5.3 and other work on test input generation for DNNs, including [21, 27]. Their ability to find adversarial examples is the usual metric for their utility. However, we emphasise that *the ultimate aim of coverage criteria is to quantify the coverage of functional features.*

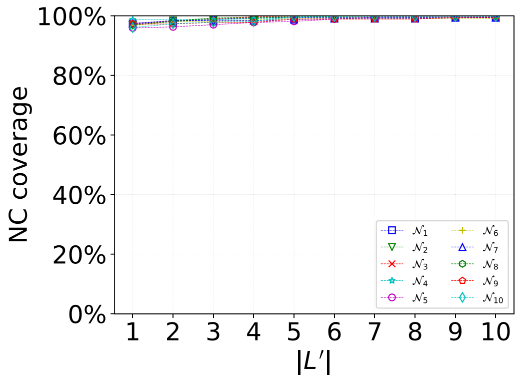
6.5 Test Data Variety as a Proxy for Functional Coverage

We have shown that our proposed test coverage criteria are a good indicator for the existence of adversarial examples. It is well-known, however, that test suites that focus on adversarial examples are not effective for assessing functional properties of DNNs. We propose a different experiment to determine how well functional coverage is approximated by our structural coverage metrics. The experiment is based on the assumption that the validation data set is a good proxy for the intent of the person who is training the DNN. We also assume that a data label corresponds to a "functional feature" of the DNN, and that the data labels have roughly similar complexity.

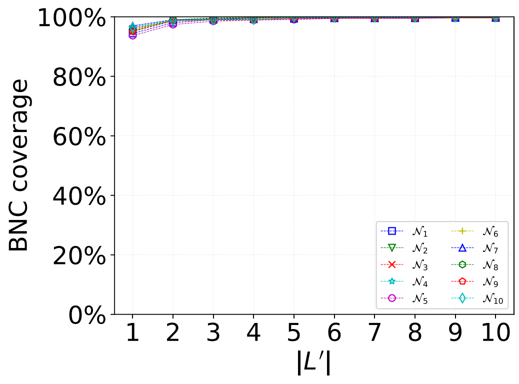
To quantify functional coverage, we first select a subset L' of the data labels L. We then hypothesize that the subset of the validation data set that contains the inputs that are labeled with one of the labels from L' yields an expected functional coverage of $|L'|/|L|$. A coverage metric that is highly correlated with this number is a good proxy for functional coverage.

In the case of MNIST, there are ten labels, i.e., $L = \{0, \dots, 9\}$. The validation data set consists of 10,000 images. We start with $L' = \{0\}$, and thus, we compute the structural coverage obtained using all images that are labeled with 0; we then proceed to $L' = \{0, 1\}$, i.e., all images labeled with either 0 or 1, and so on. We compare our proposed metric SS coverage (SSC) with neuron coverage (NC) and 2-multisection neuron coverage (BNC), as discussed in Section 4. While NC is similar to traditional statement coverage, BNC resembles branch coverage.

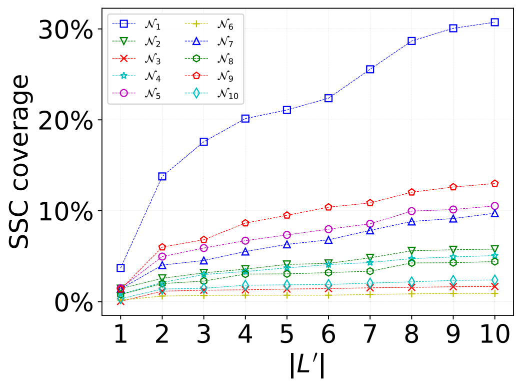
Figure 13 gives the experimental results for the MNIST DNNs in Table 3. There are several key observations.



(a) NC



(b) BNC



(c) SSC

Fig. 13. Coverage results for input data with increasing variety (from "one label only" to "ten labels") for ten MNIST DNNs.

- For all neural networks, both NC and BNC immediately report more than 90% coverage, even when given solely images that are labeled with 0. They are thus a very bad proxy for functional coverage.
- Conversely, although the slope varies for different DNNs, the level of SS coverage measured increases linearly as the variety in the test data set increases. This suggests that our structural test coverage criteria are indeed able to quantify how many test inputs are meaningfully different.
- The results in Figure 13(c) confirm that it is not trivial to achieve high SS coverage. This suggests that the test generation algorithms proposed in this paper create valuable test inputs.

7 RELATED WORK

In the following, we briefly discuss existing techniques for validating safety properties of DNNs.

Generation of Adversarial Examples for DNNs. Most existing work, e.g., [6, 13, 26, 34] applies various heuristics, generally using search 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 *means to quantify the level of confidence* (akin to testing) in the robustness of the neural network.

Testing of DNNs. There are few proposals for structural test coverage criteria for DNNs. The idea of neuron coverage is to cover both activation states of all neurons [27]. Extensions of neuron coverage are given in [21], and include criteria that check the corner values of a neuron's activation level and the activation levels of a subset of neurons in the same layer. However, the criteria in [21, 27] simply ignore the causal relationship in the DNN. In [39], 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. While in [7], coverage is enforced for finite partitions of the input space, relying on predefined sets of application-specific scenario attributes. The "Boxing Clever" technique in [3] focuses on the distribution of training data and divides the input domain into a series of representative boxes. The adversarial test generation in [36, 40] is applied to evaluate DNN-based control systems in autonomous vehicles. As shown in [19, 35], quantitative DNN coverage criteria can be applied to support the design and certification of automotive systems with deep learning components.

Automated Verification of DNNs. The safety problem of a DNN can be reduced into a constraint solving problem [9]. SAT/SMT and MILP solutions [8, 11, 15, 18, 37] have already been considered. In [16], the DNN is transformed into an equivalent hybrid system. These approaches typically only work with small networks with a few hundred hidden neurons, and overapproximation techniques [10, 22, 31] can be applied to improve the efficiency.

8 CONCLUSIONS

We have proposed a set of novel structural test coverage criteria for DNNs. Our experiments on various datasets and using two test case generation methods show promising results, indicating the applicability and effectiveness of the proposed coverage criteria. The test coverage metrics developed within this paper provide a method to obtain evidence for adversarial robustness, which is envisaged to contribute to safety cases. The criteria are also expected to provide additional insights for domain experts for evaluating the adequacy of a particular dataset as a means to describe the behavior of a DNN. In particular, our experiments suggest that they are a good proxy for functional coverage.

ACKNOWLEDGMENTS

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.

Content includes material subject to © Crown copyright (2019), Dstl. This material is licensed under the terms of the Open Government Licence except where otherwise stated. To view this licence, visit http://www.nationalarchives.gov.uk/doc/open-government-licence/version/3 or write to the Information Policy Team, The National Archives, Kew, London TW9 4DU, or email: psi@nationalarchives.gsi.gov.uk.

REFERENCES

[1] [n.d.]. Guide for Verification of Autonomous Systems. https://standards.ieee.org/project/2817.html.
[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: A system for large-scale machine learning. In *OSDI*, Vol. 16. USENIX Association, 265–283.
[3] Rob Ashmore and Matthew Hill. 2018. "Boxing clever": Practical techniques for gaining insights into training data and monitoring distribution shift. In *Computer Safety, Reliability, and Security (LNCS)*, Vol. 11094. Springer, 393–405.
[4] Rob Ashmore and Elizabeth Lennon. 2017. Progress towards the assurance of non-traditional software. In *Safety-critical Systems Symposium*.
[5] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. 2008. Speeded-up robust features (SURF). *Comput. Vis. Image Underst.* 110, 3 (June 2008), 346–359.
[6] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy (S&P)*. 39–57.
[7] Chih-Hong Cheng, Chung-Hao Huang, and Hirotoshi Yasuoka. 2018. Quantitative projection coverage for testing ML-enabled autonomous systems. In *International Symposium on Automated Technology for Verification and Analysis, ATVA (LNCS)*, Vol. 11138. Springer, 126–142.
[8] Tommaso Dreossi, Alexandre Donzé, and Sanjit A. Seshia. 2018. Compositional falsification of cyber-physical systems with machine learning components. *Journal of Automated Reasoning* (2018).
[9] Tommaso Dreossi, Shromona Ghosh, Alberto Sangiovanni-Vincentelli, and Sanjit A. Seshia. 2019. A formalization of robustness for deep neural networks. *arXiv preprint arXiv:1903.10033* (2019).
[10] Souradeep Dutta, Xin Chen, and Sriram Sankaranarayanan. 2019. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *Hybrid Systems: Computation and Control*. ACM, 157–168.
[11] Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. 2018. Output range analysis for deep feedforward neural networks. In *NASA Formal Methods Symposium (LNCS)*, Vol. 10811. Springer, 121–138.
[12] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep Learning*. Vol. 1. MIT Press.
[13] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *3rd International Conference on Learning Representations, ICLR*.
[14] Kelly Hayhurst, Dan Veerhusen, John Chilenski, and Leanna Rierson. 2001. *A Practical Tutorial on Modified Condition/Decision Coverage*. Technical Report. NASA.
[15] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. 2017. Safety verification of deep neural networks. In *Computer Aided Verification, CAV (LNCS)*, Vol. 10426. Springer, 3–29. DOI: https://doi.org/10.1007/978-3-319-63387-9_1
[16] Radoslav Ivanov, James Weimer, Rajeev Alur, George J. Pappas, and Insup Lee. 2019. Verisig: Verifying safety properties of hybrid systems with neural network controllers. In *Hybrid Systems: Computation and Control*. ACM, 169–178.
[17] Cem Kaner. 2006. Exploratory testing. In *Quality Assurance Institute Worldwide Annual Software Testing Conference*.
[18] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Computer Aided Verification, CAV (LNCS)*. Springer, 97–117.
[19] Shuyue Lan, Chao Huang, Zhilu Wang, Hengyi Liang, Wenhao Su, and Qi Zhu. 2018. Design automation for intelligent automotive systems. In *International Test Conference (ITC)*. IEEE, 1–10.
[20] David G. Lowe. 2004. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision* 60, 2 (Nov. 2004), 91–110.

[21] 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. 2018. DeepGauge: Multi-granularity testing criteria for deep learning systems. In *Automated Software Engineering (ASE)*. ACM, 120–131.

[22] Matthew Mirman, Timon Gehr, and Martin Vechev. 2018. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning, ICML*. 3575–3583.

[23] Vinod Nair and Geoffrey E. Hinton. 2010. Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning, ICML*. 807–814.

[24] Augustus Odena, Catherine Olsson, David Andersen, and Ian J. Goodfellow. 2019. TensorFuzz: Debugging neural networks with coverage-guided fuzzing. In *International Conference on Machine Learning, ICML*. PMLR, 4901–4911.

[25] Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. 2018. The building blocks of interpretability. *Distill* (2018). DOI: https://doi.org/10.23915/distill.00010

[26] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *European Symposium on Security and Privacy (EuroS&P)*. IEEE, 372–387.

[27] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. DeepXplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 1–18.

[28] RTCA. 2011. DO-178C, software considerations in airborne systems and equipment certification. (2011).

[29] SASWG. 2019. Safety assurance objectives for autonomous systems. (2019).

[30] Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR*.

[31] Xiaowu Sun, Haitham Khedr, and Yasser Shoukry. 2019. Formal verification of neural network controlled autonomous systems. In *Hybrid Systems: Computation and Control*. ACM, 147–156.

[32] Youcheng Sun, Xiaowei Huang, and Daniel Kroening. 2018. Testing deep neural networks. *CoRR* abs/1803.04792 (2018). arxiv:1803.04792 http://arxiv.org/abs/1803.04792

[33] Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. 2018. Concolic testing for deep neural networks. In *Automated Software Engineering (ASE)*. ACM, 109–119.

[34] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *ICLR*.

[35] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2017. DeepTest: Automated testing of deep-neural-network-driven autonomous cars. *arXiv preprint arXiv:1708.08559* (2017).

[36] Cumhur Erkan Tuncali, Georgios Fainekos, Hisahiro Ito, and James Kapinski. 2018. Simulation-based adversarial test generation for autonomous vehicles with machine learning components. In *Intelligent Vehicles Symposium (IV)*. IEEE, 1555–1562.

[37] Cumhur Erkan Tuncali, Hisahiro Ito, James Kapinski, and Jyotirmoy V. Deshmukh. 2018. Reasoning about safety of learning-enabled components in autonomous cyber-physical systems. In *55th Design Automation Conference (DAC)*. IEEE, 1–6.

[38] Zhou Wang, Eero P. Simoncelli, and Alan C. Bovik. 2003. Multiscale structural similarity for image quality assessment. In *Signals, Systems and Computers, Conference Record of the Thirty-Seventh Asilomar Conference on*.

[39] Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. 2018. Feature-guided black-box safety testing of deep neural networks. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS (LNCS)*, Vol. 10805. Springer, 408–426.

[40] Shakiba Yaghoubi and Georgios Fainekos. 2019. Gray-box adversarial testing for control systems with machine learning components. In *Hybrid Systems: Computation and Control*. ACM, 179–184.

[41] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. 2015. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579* (2015).

Received April 2019; revised June 2019; accepted July 2019