**DeepCT**: Tomographic Combinatorial Testing for Deep Learning Systems

Lei Ma\textsuperscript{1}, Felix Juefei-Xu\textsuperscript{2}, Minhui Xue\textsuperscript{3}, Bo Li\textsuperscript{4}, Li Li\textsuperscript{5}, Yang Liu\textsuperscript{6}, Jianjun Zhao\textsuperscript{7}

\textsuperscript{1}Harbin Institute of Technology, China \textsuperscript{2}Carnegie Mellon University, USA \textsuperscript{3}Macquarie University, Australia \textsuperscript{4}University of Illinois at Urbana–Champaign, USA \textsuperscript{5}Monash University, Australia \textsuperscript{6}Nanyang Technological University, Singapore \textsuperscript{7}Kyushu University, Japan

**Abstract**—Deep learning (DL) has achieved remarkable progress over the past decade and has been widely applied to many industry domains. However, the robustness of DL systems recently becomes great concerns, where minor perturbation on the input might cause the DL malfunction. These robustness issues could potentially result in severe consequences when a DL system is deployed to safety-critical applications and hinder the real-world deployment of DL systems. Testing techniques enable the robustness evaluation and vulnerable issue detection of a DL system at an early stage. The main challenge of testing a DL system attributes to the high dimensionality of its inputs and large internal latent feature space, which makes testing each state almost impossible. For traditional software, combinatorial testing (CT) is an effective testing technique to balance the testing exploration effort and defect detection capabilities. In this paper, we perform an exploratory study of CT on DL systems. We propose a set of combinatorial testing criteria specialized for DL systems, as well as a CT coverage guided test generation technique. Our evaluation demonstrates that CT provides a promising avenue for testing DL systems.

**Index Terms**—Deep learning, combinatorial testing, robustness

I. INTRODUCTION

The Deep learning (DL) system has achieved tremendous progress over the past decade and become the state-of-the-art technique for many cutting-edge intelligent applications. However, recent studies reveal that the robustness of DL systems is a big concern. A DL system that obtains a high prediction accuracy could still be vulnerable against adversarial attacks with minor perturbations on inputs [1]. Given that more and more safety- and security-sensitive applications start to adopt DL, deploying DL without thorough testing can potentially lead to severe consequences. Although it is highly desirable to systematically verify and provide formal guarantees on the safety and robustness of a DL system, the current attempt shows that the verification of DL systems [2] is still at an early stage and could be exceptionally challenging due to the huge runtime state space.

Software testing is widely adopted in traditional software industry due to its scalability, effectiveness, and usefulness for defect and robustness issue detection. Software testing has recently started to be applied for defect detection of deep neural networks (DNNs) [3]–[8]. Combinatorial testing (CT) is an effective testing technique that systematically explores the input configuration combinations, which can well trade off the defect detection capability and huge testing space searching effort. CT has been successfully applied to traditional configurable software systems [9]. To minimize the test suite size while obtaining desired CT coverage, CT often transforms the test generation into a constraint solving problem. The ability to handle constraints becomes crucial for real-world applications since most of the real-world systems are subjected to constraints involving multiple parameters and configurations. Hence, research on CT [10] has experienced a shift to the constrained CT which breaks down as meta-heuristic approaches [11], SAT-based approaches [12], and greedy approaches [13]. In this paper, we take the first step towards exploring whether CT is helpful for testing DL systems. We propose a set of CT coverage criteria for DL systems, as well as a CT coverage guided test generation technique. Our initial evaluation results demonstrate that CT is indeed a promising direction for testing DL systems.

II. BACKGROUND AND MOTIVATION

To better motivate this work, we discuss the intentions of this paper from the following two perspectives:

- **Per\textsubscript{1}**: Why ‘tomographic’, \textit{i.e.}, CT focuses on more intensive testing within each layer?
- **Per\textsubscript{2}**: Why ‘combinatorial’, \textit{i.e.}, CT systematically examines the interactions among neurons within each layer?

For Per\textsubscript{1}, we rely on some observations from the viewpoint of DNN formulation and properties, which allow us to arrive at a testing method that is tailored towards being tomographic. Let us take a convolutional neural network (CNN) for image processing as an example,\textsuperscript{1} and the discussion followed can be easily generalized to more general deep learning systems composed of feed-forward DNNs and recurrent neural networks. Figure 1 shows an abstraction of a particular CNN model with

---

\textsuperscript{1}CNN is commonly used for image processing, which is a special case of feed-forward deep neural network, with local receptive field and weight sharing.

![Figure 1](image-url)
multiple layers connecting input features and output features. Let \( f_l \) be the \( l \)-th image feature with \( \dim(f_l) = n \times 1 \) and \( g^{(k)}_l \) be the \( l \)-th feature map with \( \dim(g^{(k)}_l) = n^{(k)}_l \times 1 \). The convolutional layer can be mathematically represented by:

\[
g^{(k)}_l = \xi \left( \sum_{l'=1}^{q_{k-1}} W^{(k)}_{l,l'} \ast \xi \left( \sum_{l'=1}^{q_{k-2}} W^{(k-1)}_{l,l'} \ast \xi \left( \ldots \ast f_l \right) \right) \right),
\]

where \( \xi \) is the activation function (e.g., ReLU, followed by a pooling operator, such as \( g^{(k)}_l = \| g^{(k-1)}_l(x') : x' \in N(x) \|_p, p = 1, 2, \) or \( \infty \)).

For CNNs, a major assumption is that the data (image, videos, audio) are compositional with patterns that are: local, stationary, and most importantly, multi-scale (hierarchical). CNNs leverage the compositionality structure of the data and extract compositional features and feed them to the subsequent classifier, recommender, etc., in an end-to-end fashion. The design of CNNs (and DNNs in general) is entirely driven by these ‘rules’ found in the data patterns. Let us take the multi-scale property as an example, simple structures are combined to compose slightly more abstract structures, and so on, in a hierarchical and layer-by-layer way. Inspired by brain visual primary cortex [14], features learned by CNNs become increasingly more complex at deeper layers [15]. This is especially interesting because weights in various layers of a CNN are structured, hierarchical, and play different roles in the entire DL system. At any hidden layer, the neurons will consolidate all the previous information and pass along to the subsequent layers as input data with higher levels of abstraction. This is also confirmed by visualizing what the neurons learn at each layer [15], from simple combinations of corners and edges, to rich textures and 3D structure, to high-level structures like object parts.

Therefore, to test the robustness of the DNN model, it could be a good practice to test it layer by layer and systematically explore its latent feature space just like a computed tomography scan, since different layers of the DNN function at different scales. Hence, ‘tomographic’.

As for \( Pr_2 \), there are several interesting aspects to discuss. First, if we zoom into two adjacent layers in a DNN, all the neurons only interact (in the forward pass) with neurons from the next layer, and during back-propagation, all the neurons only interact (in the backward pass) with neurons from the previous layer. We can treat this local two-layer structure as an undirected bipartite graphical model, where no interactions happen within that layer. This is analogous to what ‘restricted’ refers to in the restricted Boltzmann machine. Figure 2 shows interactions between two adjacent layers using standard convolution, two variants (locally connected convolution and tiled convolution), as well as the fully connected version which is essentially the DNN case. It is obvious that there are no neural interactions within each layer.

Previous DL testing methods [3], [16] mostly focus on the measurement of the single-neuron behavior of the entire network at a time, and the coverage of a particular neuron illustrates one dimension of the entire landscape, without monitoring the interactions between neurons. In DNNs, although there are no direct (tangible) interactions between the logic units (neurons) within each computational layer, there could be logical (intangible) interactions between the logic units, where neurons of current layer altogether decide the logic of neurons of its next layer. We want to capture and examine these intangible interactions among neurons of each layer using CT. Hence, ‘combinatorial’.

Second, covering the output space of a layer of neurons (i.e., the testing input feature space of its next layer) in DL is challenging because (1) the number of output values of a neuron is often huge (e.g., infinite for continuous case), (2) neuron combination exponentially grows as neuron size increases, causing exhaustive enumeration infeasible. To sidestep, we discretize the space of the output values and only observe whether a neuron is activated. The effective testing of neurons’ combinatorial behavior instead of an exhaustive strategy is where CT comes into play. We adopt CT to systematically test diverse neuron interactions within each layer to uncover defects while reducing the number of test inputs that have to be executed. Due to the aforementioned discussions, we propose to perform tomographic CT to slice up the DL system and examine layer-wise neural behavior as depicted by Figure 1.

The contributions of the paper are summarized as follows:

- We propose a set of combinatorial testing criteria specialized for DL systems.
- We also propose a general CT coverage guided test generation technique for DL systems.
- We perform studies on DeepCT to demonstrate the potential usefulness of CT for DL testing.

To the best of our knowledge, this is the first work to explore the feasibility and usefulness of CT for DL systems.

III. COMBINATORIAL TESTING CRITERIA

We use \( N = \{ n_1, n_2, \ldots \} \) and \( T = \{ x_1, x_2, \ldots \} \) to represent the set of neurons and test inputs of a DL system, respectively. Let \( \phi(n, x) \) be a function that returns the output value of a neuron \( n \in N \) given a test input \( x \in T \). For a DNN with \( m \) layers, we use \( L_i \) to denote the set of all neurons on its \( i \)-th layer \( l_i \) (1 \( \leq i \leq m \)). Our CT coverage criteria for DNNs are motivated by the combinatorial coverage metrics in traditional software testing, and the ways that a DNN processes an input through transformations layer by layer. Similar to previous work [3], [16], we consider the neuron \( n \) to be activated (resp. deactivated) given \( x \) if \( \phi(n, x) \in (0, \infty) \) (resp. \( \phi(n, x) \in (-\infty, 0] \)). We use \( A(n, x) \in \{ 0, 1 \} \) to represent the
activation status of a neuron $n_i$ given $x$, where 1 corresponds to activation and 0 deactivation. To test the functionality of a layer, we first introduce the neuron-activation configuration.

**Definition III.1 (Neuron-activation configuration).** For a set of neurons $M = \{n_1, n_2, \ldots, n_k\} \subseteq L_i$ of the $i$-th layer $L_i$, a neuron-activation configuration of $M$ is a tuple $c = (b_1, b_2, \ldots, b_k)$, where $b_i = \{0, 1\}$. A neuron-activation configuration $c = (b_1, b_2, \ldots, b_k)$ is undefined if there exists at least a test input $x \in T$, such that $b_i = A(n_i, x)$ for all $i$, where $1 \leq i \leq k$.

Given a test set $T$ and the neurons $L_i$ of the $i$-th layer, we use $\Theta(t, L_i)$ to denote the set of all $t$-way combinations of neurons in $L_i$. Each element $\theta$ (i.e., $\theta \in \Theta(t, L_i)$) is a set of $t$ neurons, with a total of $2^t$ neuron-activation configurations.

Furthermore, we use $\Theta_{\text{full}}(t, L_i, T)$ (i.e., $\Theta_{\text{full}}(t, L_i, T) \subseteq \Theta(t, L_i)$) to represent those $t$-way combinations of neurons of $L_i$, where all the neuron-activation configurations of $\theta_{\text{full}} \in \Theta_{\text{full}}(t, L_i, T)$ are fully covered by $T$. For a $t$-way neuron combination $\theta \in \Theta(t, L_i)$, we use $C(t, \theta, T)$ to represent the set of neuron-activation configurations of $\theta$ covered by $T$. Next, we define the $t$-way combinatorial testing criteria.

**Definition III.2 ($t$-way combination sparse coverage).** We define the $t$-way combination sparse coverage as the percentage of $t$-way neuron combinations in $L_i$, of which all the neuron-activation configurations are covered by $T$,

$$\text{SparseCov}(t, L_i, T) = \frac{|\{\theta \in \Theta(t, L_i) | \theta \in \Theta_{\text{full}}(t, L_i, T)\}|}{|\Theta(t, L_i)|}$$

**Example III.1.** Let $L_i = \{n_1, n_2, n_3, n_4\}$ be a set of neurons in the same layer. Each row in Figure 3 corresponds to the neuron activation status given a test set $T$, denoted by $A(L_i, x)$. There are in total six $2$-way combinations of neurons in $L_i$, $\{n_1, n_2\}$, $\{n_1, n_3\}$, $\{n_1, n_4\}$, $\{n_2, n_3\}$, $\{n_2, n_4\}$ and $\{n_3, n_4\}$. Each $2$-way combination has four neuron-activation configurations $(0, 0)$, $(0, 1)$, $(1, 1)$ and $(1, 0)$. Among the six $2$-way combinations, only the full neuron-activation configurations of $\{n_1, n_2\}$, $\{n_1, n_4\}$, $\{n_2, n_3\}$ and $\{n_3, n_4\}$ are covered by $T$. Therefore, the $2$-way combination sparse coverage for $T$ on $L_i$ is $66\%$.

Since the $t$-way combination sparse coverage cannot take the coverage within each combination of neurons into account, next we introduce the $t$-way combination dense coverage.

**Definition III.3 ($t$-way combination dense coverage).** For a set of test inputs $T$ and a set of neurons $L_i$, the $t$-way combination dense coverage is defined as below,

$$\text{DenseCov}(t, L_i, T) = \frac{\sum_{\theta \in \Theta(t, L_i)} |C(t, \theta, T)|}{2^t |\Theta(t, L_i)|}$$

**Example III.2.** Consider the neuron activation status of $L_i$ given test set $T$ in Figure 3. Since there are six $2$-way combinations of neurons in $L_i$, and each has four neuron-activation configurations, there are $24$ neuron-activation configurations in total. The test set $T$ can cover $20$ configurations and the neuron-activation configurations that are not covered are $\{n_1, n_3\} = (0, 1), \{n_1, n_3\} = (1, 0), \{n_2, n_4\} = (0, 1)$ and $\{n_2, n_3\} = (1, 0)$. Therefore, the $2$-way combination sparse coverage for $T$ is $66.6\%$.

**Definition III.4 ($p, t$-completeness).** For a set of neurons $L_i$ and test inputs $T$, $(p, t)$-completeness is defined as the proportion of those $\theta \in \Theta(t, L_i)$, where the covered $t$-way neuron activation configuration ratio of $\theta$ is at least $p$ (i.e., $C(t, \theta, T)/2^t \geq p$).

**Example III.3.** We again take Figure 3 as an example. For $2$-way combinations of neurons, the covered $2$-way configuration ratio for $\{n_1, n_2\}$, $\{n_1, n_4\}$, $\{n_2, n_3\}$ and $\{n_3, n_4\}$ are $100\%$ and $50\%$ for $\{n_1, n_3\}$ and $\{n_2, n_4\}$. According to the Definition III.4, the $(0.5, 2)$-completeness of $L_i$ is $100\%$ and the $(1, 2)$-completeness of $L_i$ is $66.6\%$.

Although the combinatorial testing criteria defined above consider neurons for a layer, they could be easily generalized to the whole neural network level layer by layer. We leverage these criteria to evaluate the testing effectiveness from the perspective of CT and to further guide test generation.

**IV. CT Robustness Testing of Deep Learning**

Our proposed CT coverage criteria are general for both test suite evaluation and test generation guidance. In this study, we consider a typical problem of using CT criteria to test the adversarial robustness of DNNs designed for the general-purpose image classification [17]. Given an input $x$ that can be correctly classified by a DNN, the adversarial robustness property is concerned with whether there exists another input $x'$ close to $x$, with respect to some distance metrics (e.g., $L_\infty$-norm), where $x$ and $x'$ are classified to different classes by the DNN. Such an input $x'$, once exists, is called an adversarial example of $x$ and the DNN is not locally robust at $x$.

Let $C(x)$ denote the class to which $x$ is classified by DNNs. Formally, a DNN is $d$-locally-robust at an input $x$ w.r.t. a distance parameter $d$ if we have the following [17]:

$$\forall x': \|x' - x\| \leq d \Rightarrow C(x) = C(x')$$

Adversarial robustness could be analyzed by checking the local robustness property. Therefore, the rest of this paper focuses on testing the local robustness of DNNs.

![Fig: 3: Example of the neuron activation status of $L_i$ in layer $l_i$ (that contains 4 neurons) given test inputs $T$.](image-url)
We have implemented DeepCT, a DL tomographic combinatorial testing framework that performs automated test generation for DNNs based on Keras (ver.2.1.3) and Tensorflow (ver.1.5.0). The current version of DeepCT provides a LP constraint solving based test generator, which we use to investigate whether CT and our proposed criteria are useful for testing DNNs.

In particular, we mainly investigate whether DeepCT and our criteria are useful for adversarial example detection by local robustness analysis. We use the publicly available dataset MNIST and two pre-trained DNN models [4] that achieve competitive prediction accuracy. The two studied DNNs contain 3 (64*32*64 with 55,082 parameters) and 5 (84*42*64*42*84 with 79,454) fully-connected hidden layers, and obtain 99.965%, 99.872% training accuracy, and 97.63%, 97.51% test accuracy respectively. For the DNNs' local robustness analysis, we randomly seed 1,000 tests from MNIST accompanied test set as the study subject, which can be correctly handled by our studied DNNs.

A. Random Testing

Although previous work [4] advocated that random testing (RT) is ineffective in detecting the adversarial robustness issues of DNNs, we believe that RT is easy to use and scalable, which is worth a first shot before further in-depth analysis. Therefore, our first step performs random test generation to analyze the robustness of two studied DNNs (i.e., $DNN_1$ and $DNN_2$) on the 1,000 seeded tests. To be specific, we randomly generate 10,000 tests for each seeded test and analyze whether robustness issues could be detected. The experiment results show that the RT is already able to detect robustness issues on 194 seeded tests on $DNN_1$ and 178 on $DNN_2$ with a total of 266 unique issues, 106 of which are shared issues on both $DNN_1$ and $DNN_2$. This is consistent with our intuition that the robustness of a DNN on handling different test input could be different. A test input near a DNN’s decision boundary could trigger robustness issues more easily. Our experimental results confirm this observation and indicate that RT could be useful to detect the fragile test input as the first attempt.

B. DeepCT

For the 1,000 seeded tests, we filter out those 266 tests whose robustness issues can already be detected by random testing. For the remaining 734 tests, we randomly sample 50 tests for further analysis on $d$-locally-robustness by DeepCT (where $d = 0.15$, and we use 2-way defined CT criteria, also see Algorithm 1) in line with the corresponding tests generated by random testing. DeepCT incrementally generates tests to cover CT coverage targets layer by layer. Table I summarizes the averaged results. Columns 4-7 show the obtained coverage of 2-way sparse, 2-way dense, (0,5,2)-completeness, and (0.75,2)-completeness coverage of DeepCT under the corresponding criterion setting. Column 8 gives the accumulated total number of generated tests and Column 9 shows the corresponding detected adversarial tests ratio.

Overall, for both studied DNNs, random testing achieves fairly low coverage of all the evaluated criteria and DeepCT achieves much higher coverage although with a similar number

$4$Each of 784 pixels of the test image is normalized to range $[0, 1]$. In this study, we allow the random perturbation of each pixel within range $[-0.15, 0.15]$. 

of tests. In addition, DeepCT increases the coverage as more layers of a DNN is tested with more tests generated. We could also see that, DeepCT could detect 0.15-local-robustness issues on all of the 50 studied tests for both DNNs with a reasonable number of tests. To be specific, Table I shows that random testing achieves only 2.28% and 1.18% 2-way sparse coverage on two studied DNNs. Compared with (0.5,2)-completeness coverage, (0.75,2)-completeness coverage is also much lower, indicating that random testing does not deeply cover many of the neuron activation configurations of 2-way neuron combinations. In comparison, DeepCT obtains 60.27% and 46.96% 2-way sparse coverage even only the first hidden layer is analyzed, with a reasonable test suite size while already being able to detect some adversarial examples for all seeded tests. As described in Algorithm 1, when DeepCT analyzes the second hidden layer \( l_2 \), it first analyzes the coverage obtained by tests generated by all previous layers, and generates tests to cover the uncovered target on layer \( l_2 \). In the two studied DNNs, after generating tests for layer \( l_2 \), 2-way sparse coverage of the whole network increases by 16.67% and 21.95%, respectively. Similar coverage improvement could also be observed by other coverage criteria. For example, the obtained (0.75,2) coverage indicates that more than 80% of the 2-way neuron interactions are mostly covered deeply. When the first three layers of the two studied DNNs are analyzed, the 2-way sparse coverage reaches 93.62% and 97.15%, respectively, with many of the adversarial examples detected. For DNN2, only about 220 new tests in total are created when analyzing layers \( l_4 \) and \( l_5 \). As for the detected adversarial examples, adversarial ratio gaps of \( l_2 \) and \( l_3 \) are much larger than other layers for both DNNs. This indicates that the different layers might contribute differently to detect adversarial examples by CT, and some layers might need to be more intensively tested.

VI. CONCLUSION

Combinatorial testing is a well-established and useful technique in traditional software testing. Rather than exhaustively searching all the combinations of input space, CT focuses on testing the interactions of inputs, aiming to reduce the test suite size while obtaining satisfiable defect detection abilities. This paper initiates the first study to explore the usefulness of CT for testing DL systems. Our results show that CT provides a promising avenue for testing DL systems. Our future work would perform more in-depth investigation on the effect of \( t \) in \( t \)-way CT for adversarial example detection, and incorporate more scalable CT test generators for real-world DL systems.

ACKNOWLEDGEMENTS

This work was partially supported by 973 Program (No. 2015CB352203), Fundamental Research Funds for the Central Universities (No. A015710000816) of China, and JSPS KAKENHI Grant 18H04097. We gratefully acknowledge the support of NVIDIA AI Tech Center (NVAITC) to our research.

REFERENCES