DeepCT: Tomographic Combinatorial Testing for Deep Learning Systems

Lei Ma[1], Felix Juefei-Xu[2], Minhui Xue[3], Bo Li[4], Li Li[5], Yang Liu[6], Jianjun Zhao[7]
[1]Harbin Institute of Technology, China [2]Carnegie Mellon University, USA [3]Macquarie University, Australia
[4]University of Illinois at Urbana–Champaign, USA [5]Monash University, Australia
[6]Nanyang Technological University, Singapore [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.

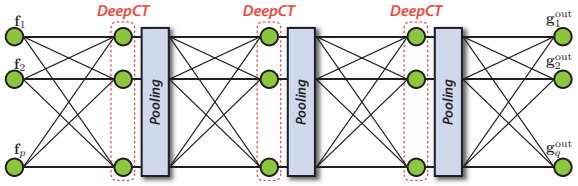


Fig. 1: For CNNs, the features are composed of multiple convolutional and pooling layers. DeepCT examines the interactions among neurons within each layer, just like a computed tomography scan.

II. Background and Motivation

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

- Per_1: Why 'tomographic', *i.e.*, CT focuses on more intensive testing within each layer?
- Per_2: Why 'combinatorial', *i.e.*, CT systematically examines the interactions among neurons within each layer?

For Per_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,[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

[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.

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

$$\mathbf{g}_l^{(k)} = \xi\left(\sum_{l'=1}^{q_{k-1}} \mathbf{W}_{l,l'}^{(k)} \star \xi\left(\sum_{l'=1}^{q_{k-2}} \mathbf{W}_{l,l'}^{(k-1)} \star \xi\left(\cdots \mathbf{f}_{l'}\right)\right)\right),$$

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

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 Per_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

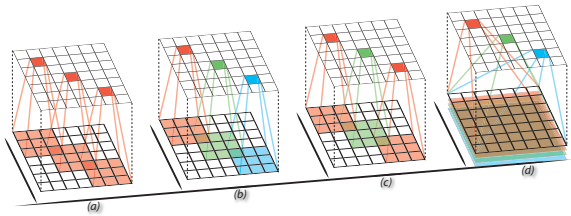


Fig. 2: Interactions between two adjacent layers for (a) standard convolution, (b) locally connected convolution, (c) tiled convolution, and finally (d) fully connected case (DNN).

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 $T = \{\mathbf{x}_1, \mathbf{x}_2, \ldots\}$ to represent the set of neurons and test inputs of a DL system, respectively. Let $\phi(n, \mathbf{x})$ be a function that returns the output value of a neuron $n \in N$ given a test input $\mathbf{x} \in T$. 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 $\mathbf{x}$ if $\phi(n, \mathbf{x}) \in (0, \infty)$ (resp. $\phi(n, \mathbf{x}) \in (-\infty, 0]$). We use $A(n_i, \mathbf{x}) \in \{0, 1\}$ to represent the

activation status of a neuron n_i given $\mathbf{x}$, 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 $c = (b_1, b_2, ..., b_k)$, where $b_i = \{0, 1\}$.[2] A neuron-activation configuration $c = (b_1, b_2, ..., b_k)$ is covered by T if there exists at least a test input $\mathbf{x} \in T$, such that $b_i = A(n_i, \mathbf{x})$ 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 θ (*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 $\Theta_{full}(t, L_i, T)$ (*i.e.*, $\Theta_{full}(t, L_i, T) \subseteq \Theta(t, L_i)$) to represent those t-way combinations of neurons of L_i, where all the neuron-activation configurations of $\theta_{full} \in \Theta_{full}(t, L_i, T)$ 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 θ 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,

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

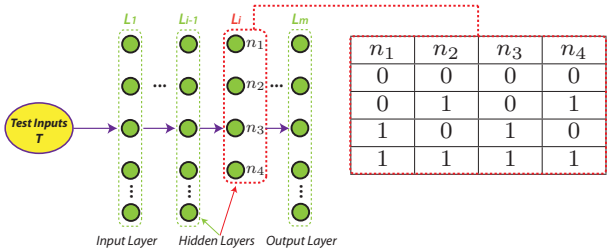
Example III.1. *Let $L_i = \{n_1, n_2, n_3, n_4\}$ be a set of neurons in the same layer. Each row j in Figure 3 corresponds to the neuron activation status given a test of T, denoted by $A(L_i, \boldsymbol{x}_j)$. There are in total six 2-way combinations of neurons in L_i, $\{n_1, n_2\}$, $\{n_1, n_3\}$, $\{n_1, n_4\}$, $\{n_2, n_3\}$, $\{n_2, n_4\}$ and $\{n_3, n_4\}$. Each 2-way combination has four neuron-activation configurations $(0, 0)$, $(0, 1)$, $(1, 0)$ and $(1, 1)$. 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 66.6%.*

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 $\{n_2, n_4\} = (1, 0)$. Therefore, the 2-way combination dense coverage for T is 83.3%. Recall that the 2-way combination sparse coverage for T is 66.6%.*



n_1	n_2	n_3	n_4
0	0	0	0
0	1	0	1
1	0	1	0
1	1	1	1

Fig. 3: Example of the neuron activation status of L_i in layer l_i (that contains 4 neurons) given test inputs T.

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 θ 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 for 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 $\mathbf{x}$ that can be correctly classified by a DNN, the adversarial robustness property is concerned with whether there exists another input $\mathbf{x}'$ close to $\mathbf{x}$, with respect to some distance metrics (*e.g.*, L_∞-norm), where $\mathbf{x}$ and $\mathbf{x}'$ are classified to different classes by the DNN. Such an input $\mathbf{x}'$, once exists, is called an adversarial example of $\mathbf{x}$ and the DNN is not locally robust at $\mathbf{x}$.

Let $\mathcal{C}(\mathbf{x})$ denote the class to which $\mathbf{x}$ is classified by DNNs. Formally, a DNN is d-locally-robust at an input $\mathbf{x}$ w.r.t. a distance parameter d iff we have the following [17][3]:

$$\forall \mathbf{x}' : \|\mathbf{x}' - \mathbf{x}\| \leq d \Rightarrow \mathcal{C}(\mathbf{x}) = \mathcal{C}(\mathbf{x}')$$

[2]Note that k is not necessarily the number of neurons of the layer L_i. In the case of t-way combination of L_i, k equals to t.

[3]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.

Algorithm 1: DeepCT Test Generation

INPUT: DNN N, t-way, CT Criteria CTC, seeding Tests T_s
OUTPUT: Passed Test Suite T, Adversarial Test Suite T'

1: $T \leftarrow \{\}, T' \leftarrow \{\}$
2: CT_table←initialize_CT_coverage_table(T_s, t-way, CTC)
3: **for** $t \in T_s$ **do**
4: $T_{work} \leftarrow \{\}$
5: **for** each layer neuron set L_i in N **do**
6: CT_table ← update_CT_coverage(CT_table, T_{work}, CTC)
7: CT_targets←calculate_CT_targets(CT_table, T_{work}, L_i, CTC)
8: **while** CT_targets $\neq \emptyset \wedge$ time budget exists **do**
9: ct_j ←random_select(CT_targets)
10: gen_tests←TestGen(t, ct_j)
11: covered_targets ← cal_cov_targets(CT_table, T_{work}, gen_tests, CTC)
12: CT_targets←CT_targets - covered_targs
13: update_TestSuite(gen_tests, T_{work}, T, T')
14: **return** T, T'

One approach to analyze local robustness is through random testing. However, random testing is often ineffective even if a large number of tests are generated [4]. To systematically generate tests to detect local robustness issues of a given input, we propose a combinatorial testing technique for test suite generation of a DL system. Algorithm 1 shows the details of our CT coverage guided test generation. Given a seeded test set, t-way, and CT criterion as input, the CT coverage table of the whole DNN is first initialized (see Line 1-2). Then, the test generation iteration starts for each seeding test guided by the CT coverage layer by layer. For each candidate layer, the coverage is analyzed on the generated tests so far. The coverage table is updated and the uncovered CT targets of a layer l_i are calculated (Line 6-7). After randomly selecting a target ct_j, we try to generate tests to cover ct_j and analyze all the coverage targets they reach. Note that a test case might cover multiple CT targets in t-way testing, and the generated tests might not cover any CT targets (Line 9-12). Before attempting on the next CT target, we check whether the generated tests contain adversarial examples and update the generated test suite for T, T', and the test working set T_{work} accordingly (Line 13). The test generation iteration continues until CT coverage targets are covered or the time limit hits.

Our test generation technique to cover the specific CT coverage target is general without assumption on specific types of DNN structures (*e.g.*, activation functions) or test generation methods (*e.g.*, search-based testing [18], guided random testing [19], symbolic constraint solving based testing [20]). To demonstrate the CT coverage guided test generation is helpful for detecting adversarial examples, this study assumes that DNN uses the popular `ReLU` activation function and adopts the constraint solving based (*i.e.*, the Cplex solver) test generator [4]. In particular, a CT coverage target is encoded as the linear constraints with the object to minimize the L_∞-norm perturbation distance on a seeding input.

V. Evaluation

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.[4] 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 $[-0.15, 0.15]$.

TABLE I: The obtained CT coverage and detected adversarial issues by random testing and DeepCT. The coverage is calculated by considering the 2-way CT targets of all hidden layers of the DNN instead of a single layer.

	Testing	2-way Combinatorial Testing Coverage (%)				Accu.	Adv.
	Method	SparseCov	DenseCov	(0.5,2)	(0.75,2)	Tests	Per.(%)
DNN_1	Random	2.28	34.95	33.75	3.75	10,000	0.00
	CT l_1	60.27	81.56	95.01	70.98	4,073	0.29
	CT l_2	76.94	91.98	99.67	91.30	6,768	2.17
	CT l_3	93.62	98.23	100.00	99.32	8,032	9.91
DNN_2	Random	1.18	32.56	26.98	2.10	10,000	0.00
	CT l_1	46.96	75.10	91.95	61.50	8,547	1.87
	CT l_2	68.91	87.52	98.64	82.55	11,573	3.53
	CT l_3	97.15	99.05	100.0	99.03	13,129	8.84
	CT l_4	97.41	99.11	100.0	99.03	13,217	9.35
	CT l_5	97.81	99.21	100.0	99.03	13,351	9.98

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% 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 DNN_2, 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. AUGA5710000816) of China, and JSPS KAKENHI Grant 18H04097. We gratefully acknowledge the support of NVIDIA AI Tech Center (NVAITC) to our research.

References

[1] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *ICLR*, 2015.

[2] W. Xiang, P. Musau, A. A. Wild, D. Manzanas Lopez, N. Hamilton, X. Yang, J. Rosenfeld, and T. T. Johnson, "Verification for Machine Learning, Autonomy, and Neural Networks Survey," *ArXiv e-prints*, p. arXiv:1810.01989, Oct. 2018.

[3] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 1–18.

[4] Y. Sun, X. Huang, and D. Kroening, "Testing Deep Neural Networks," *ArXiv e-prints*, Mar. 2018.

[5] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu, J. Zhao, and Y. Wang, "Deepgauge: Multi-granularity testing criteria for deep learning systems," in *The 33rd IEEE/ACM International Conference on Auto. Software Engg. (ASE)*, 2018.

[6] L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. Juefei-Xu, C. Xie, L. Li, Y. Liu, J. Zhao, and Y. Wang, "DeepMutation: Mutation Testing of Deep Learning Systems," in *The 29th IEEE International Symposium on Software Reliability Engineering (ISSRE)*, 2018.

[7] X. Xie, L. Ma, F. Juefei-Xu, H. Chen, M. Xue, B. Li, Y. Liu, J. Zhao, J. Yin, and S. See, "DeepHunter: Hunting Deep Neural Network Defects via Coverage-Guided Fuzzing," *arXiv preprint arXiv:1809.01266*, 2018.

[8] X. Du, X. Xie, Y. Li, L. Ma, J. Zhao, and Y. Liu, "Deepcruiser: Automated guided testing for stateful deep learning systems," *arXiv preprint arXiv:1812.05339*, 2018.

[9] C. Nie and H. Leung, "A survey of combinatorial testing," *ACM Computing Surveys (CSUR)*, vol. 43, no. 2, p. 11, 2011.

[10] B. S. Ahmed, L. M. Gambardella, and et al., "Handling constraints in combinatorial interaction testing in the presence of multi objective particle swarm and multithreading," *IST*, vol. 86, pp. 20–36, 2017.

[11] Y. Jia, M. B. Cohen, M. Harman, and J. Petke, "Learning combinatorial interaction test generation strategies using hyperheuristic search," in *IEEE/ACM 37th Intl. Conf. on Softw. Egg. (ICSE)*, 2015, pp. 540–550.

[12] A. Yamada, T. Kitamura, C. Artho, E.-H. Choi, Y. Oiwa, and A. Biere, "Optimization of combinatorial testing by incremental sat solving," in *The 8th Intl. Conf. on Softw. Test. Verif. & Valid. (ICST)*, 2015, pp. 1–10.

[13] J. Czerwonka, "Pairwise testing in real world," in *24th Pacific Northwest Software Quality Conference*, vol. 200. Citeseer, 2006.

[14] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of physiology*, vol. 160, no. 1, pp. 106–154, 1962.

[15] C. Olah, A. Satyanarayan, I. Johnson, S. Carter, L. Schubert, K. Ye, and A. Mordvintsev, "The building blocks of interpretability," *Distill*, 2018.

[16] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proc. of the 40th International Conference on Software Engineering*, 2018, pp. 303–314.

[17] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," *CoRR*, vol. abs/1702.01135, 2017.

[18] P. McMinn, "Search-based software testing: Past, present and future," in *ICST'11*, 2011, pp. 153–163.

[19] L. Ma, C. Artho, C. Zhang, H. Sato, J. Gmeiner, and R. Ramler, "Grt: Program-analysis-guided random testing (t)," in *The 30th IEEE/ACM Intl. Conf. on Auto. Softw. Engineering (ASE)*, 2015, pp. 212–223.

[20] R. Baldoni, E. Coppa, D. C. D'Elia, C. Demetrescu, and I. Finocchi, "A survey of symbolic execution techniques," *ACM Comput. Surv.*, vol. 51, no. 3, 2018.