2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)

DeepConcolic: Testing and Debugging Deep Neural Networks

Youcheng Sun*, Xiaowei Huang†, Daniel Kroening*, James Sharp‡, Matthew Hill‡, Rob Ashmore‡

*University of Oxford, UK
{youcheng.sun, kroening}@cs.ox.ac.uk
†University of Liverpool, UK
xiaowei.huang@liverpool.ac.uk
‡Defence Science and Technology Laboratory (Dstl)
{jsharp1, mhill2, rdashmore}@dstl.gov.uk

***Abstract*—Deep neural networks (DNNs) have been deployed in a wide range of applications. We introduce a DNN testing and debugging tool, called DeepConcolic, which is able to detect errors with sufficient rigour so as to be applicable to the testing of DNNs in safety-related applications. DeepConcolic is the first tool that implements a concolic testing technique for DNNs, and the first testing tool that provides users with the functionality of investigating particular parts of a DNN. The tool has been made publicly available and a demo video can be found at https://youtu.be/rliynbhoNLM.**

I. Introduction

The wide-scale deployment of deep neural networks (DNNs) in safety-critical applications such as self-driving cars, healthcare and UAVs, increases the demand for tools that can test, validate, and ultimately certify software that implements DNNs [1]. Testing has been the primary approach for providing assurance evidence to support safety arguments, and enables certification of safety-critical software systems. Owing to their complexity, DNNs are frequently considered black boxes, as it is difficult to understand their behaviour by means of inspection. White-box methods, by contrast, exploit the structure of a program to gather assurance evidence. The application of white-box testing in a way that supports arguments about the behaviour of a system is challenging.

We consider structural coverage metrics, which are a popular exemplar of white-box testing. For instance, a test suite with 100% statement coverage exercises all statements of a program at least once. While structural coverage has no guarantee for correct functionality, it is frequently used as a proxy metric for the exhaustiveness of a test suite [2]. Structural coverage metrics such as modified condition/decision coverage (MC/DC) [3] are used as a means of assessing the exhaustiveness of a test suite in a number of high-tier safety standards in a wide range of application domains, including avionics [4]. Structural coverage criteria have been developed to examine DNNs [5], [6], and to guide the generation of test suites to support an equivalent form of testing for DNNs. Whilst structural coverage in DNNs is not a direct reflection of behavioural confidence, the proposed techniques may provide a means of understanding, when combined with other techniques, the adequacy of the testing regime.

In this paper, we introduce DeepConcolic, a tool for testing and debugging DNNs. Its test generation is guided by coverage criteria that have been adapted from statement coverage and MC/DC; it adopts a concolic analysis that examines different behaviours of DNNs, and is able to isolate potentially problematic components in the DNN. Concolic testing [6] is a hybrid approach for testing software and it has recently been extended to DNNs. More specifically, the testing process repeatedly switches between concrete execution of a DNN and its symbolic encoding to find inputs that trigger specified behaviours.

The tool can be useful for not only DNN test engineers, who can run coverage-based testing, but also DNN developers, who need to analyse the internal structure of DNNs.

A. DNNs

Formally, a (feedforward and deep) neural network [7], or DNN, is a tuple $\mathcal{N} = (L, T, \Phi)$, where $L = \{L_k \mid k \in \{1..K\}\}$ is a set of layers, $T \subseteq L \times L$ is a set of connections between layers and $\Phi = \{\phi_k \mid k \in \{2..K\}\}$ is a set of non-linear activation functions, one for each non-input layer. Each layer L_k consists of s_k *neurons*.

A layer in the DNN can be a dense (fully-connected) layer, convolutional layer, flatten layer or max-pooling layer, with activation functions being, for example, Rectified Linear Unit (ReLU), softmax, etc. Particularly, every neuron in the convolutional layer belongs to a so-called *feature map* that is defined by the kernel operation. For any input, the DNN assigns a *label* that is denoted as $\mathcal{N}(x).label$.

B. Safety in DNNs

Notably, *adversarial examples* [8], whereby two *close enough* inputs cause contradictory decisions, are one of the most prominent safety concerns to date for DNNs. Given an input x_1 that is correctly labeled by a DNN $\mathcal{N}$, another input x_2 is said to be an adversarial example, if x_1 and x_2 are close enough such that $||x_1 - x_2||_p \leq \varepsilon$ and $\mathcal{N}(x_1).label \neq \mathcal{N}(x_2).label$, where p denotes the L^p-norm distance metric and ε is sufficiently small.

2574-1934/19/$31.00 ©2019 IEEE
DOI 10.1109/ICSE-Companion.2019.00051

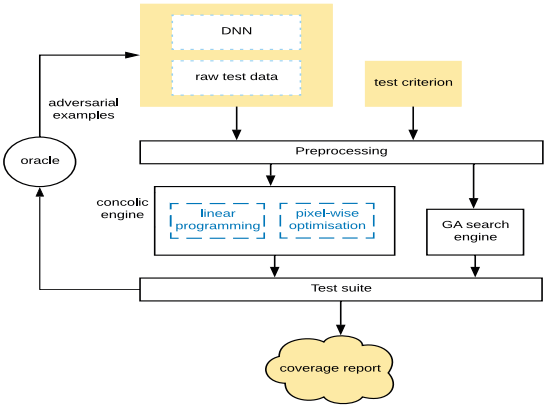


Fig. 1: The DeepConcolic tool

A number of algorithms have been proposed to find adversarial examples. However, such attack methods are not able to quantify the adequacy level (or the stop condition) for testing a DNN. This fact motivates the coverage criteria recently developed for DNNs.

Several structural coverage criteria have been designed for DNNs, including neuron coverage [9] and its extensions [10], and the MC/DC variants for DNNs [5]. Such coverage criteria aim to define test conditions that are more likely to capture adversarial behaviours in a DNN. Neuron coverage asks that, for every neuron in the network, there must exist at least one input that makes its activation value larger than some threshold; the criteria in [10] generalise the neuron coverage from a single neuron to a set of neurons. The MC/DC variants for DNNs capture the fact that a (decision) feature (a set of neurons) in a layer is directly decided by its connected (condition) features in the previous layer, and thus requires test conditions such that every condition feature must be shown regardless of its effect on the decision feature.

II. The DeepConcolic Tool

The overall architecture of the DeepConcolic tool is shown in Figure 1. In principle, given a DNN model and a coverage criterion, DeepConcolic generates a test suite and returns the coverage result. In order to generate meaningful test cases for the DNN, some input test data is also supplied as a reference for finding suitable nearby test cases.

Specifically, the tool has the following features.

a) Test criterion: DeepConcolic is a coverage-guided testing tool and it currently supports neuron coverage and the MC/DC variants for DNNs.

b) Preprocessing: The preprocessing module formats the input data and configures the back-end testing engine, i.e., the concolic engine and the gradient ascent (GA) search engine that generate test cases. Particularly, a test_objectt class is defined to encode the formatted inputs and command line configurations for running DeepConcolic. Meanwhile, the covering_layert class encapsulates each DNN layer to be covered and implements the data structures needed for the coverage criteria.

c) Concolic engine: The concolic engine is one of our two test case generation engines. It implements the algorithms in [6]. Concolic testing combines the execution of concrete inputs and a symbolic analysis technique to efficiently meet test conditions from the specified coverage criterion. There are two symbolic techniques that users can choose from: (1) the linear programming (LP) approach optimises using L^{∞}-norm that is the maximum change between every dimension of two inputs; and (2) the global optimisation approach applies to L^{0}-norm using pixel-wise manipulation (of input images). More details are provided in [6].

d) GA search engine: We also integrate the adaptive GA search algorithm in [5] for test case generation. Gradient-based search algorithms have been shown to be effective for DNN testing in several other recent works [9], [11], [10].

e) Test suite: The test suite generated from DeepConcolic consists of a set of input pairs combined with the distance between them.

f) Oracle: The oracle follows the definition of adversarial examples in Section I-B, according to which adversarial examples are picked up from the test suite. This is achieved by specifying how distant a correctly classified input may deviate before being classified differently.

g) Coverage report: The report includes the level of coverage reached for the specified criterion, together with the number of test cases generated at each step, the number of adversarial examples, the distance for each adversarial example, and some traceability information, etc.

h) Others: As a white-box testing tool, one aim of DeepConcolic is to help developers understand and diagnose the internal structure of a DNN. A handy command line option for DeepConcolic enables the specification of a particular layer, or a subset of neurons, upon which to test. Besides the coverage report, DeepConcolic also returns quantitative statistics for the robustness of the network to adversarial examples. Note that such statistics would need to be analysed in the context of the target application, and its acceptability needs to be argued within a safety argument.

In summary, we see the following advantages for using DeepConcolic. It generates test cases and adversarial examples for DNNs following the specified test conditions. Developers can use the test results to compare different DNN models, and the adversarial examples can be used to improve and re-train the DNN or develop an adversarial example mitigation strategy. Moreover, a major safety challenge for the use of DNNs is due to the lack of understanding about how a decision is made by a DNN. The DeepConcolic tool is able to test each particular component of a DNN, and this aids human analysis of the internal structures of a DNN. By understanding these structures, we improve the confidence in DNNs; this is different from providing guarantees, just as testing conventional software does not provide guarantees.

DeepConcolic is implemented in Python and is open source. It is publicly available on GitHub[1]. The only other public DNN

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

structural coverage guided testing tool is DeepXplore from [9] that is dedicated to neuron coverage. A preliminary comparison between DeepConcolic and DeepXplore can be found in [6].

A. Example usage

The DeepConcolic tool supplies the users with a list of command line options. Here we explain its usage through examples. For example, the following command

```
python deepconcolic.py --model cifar10.h5 --cifar10-data
        --outputs outs --layer-index 3 --feature-index 0
        --criterion ssc --cond-ratio 0.1
```

calls DeepConcolic to test a DNN model that is trained on the CIFAR-10 dataset, with the flag *--cifar10-data* to automatically load the input data. Subsequently, all testing results will be deposited into a specified directory *outs*. The *--layer-index* and *--feature-index* set up the layer index and feature map index to test, respectively. The options *--criterion ssc --cond-ratio* 0.1 configure the MC/DC variants for DNNs.

The following command runs DeepConcolic on a pre-trained VGG16 model. The input data is stored in a directory named *inputs*. The option *--top-classes* 5 means the top-5 accuracy will be considered for the model and *--labels labels.txt* specifies the labels for the input data.

```
python deepconcolic.py --vgg16 --data inputs
        --outputs outs --criterion ssc --cond-ratio 0.1
        --top-classes 5 --labels labels.txt
```

III. Experiments

In this section, we demonstrate the utility of DeepConcolic by applying it to a DNN of moderate size that is trained using the CIFAR-10 dataset, which contains 60,000 32x32 color images in 10 different classes (e.g., automobile). Experiments are conducted on a mobile workstation with 2.90GHz Intel Core i7, 16GB memory and NVIDIA Quadro M1200.

In the experiment, we test two individual convolutional layers (layer 2 and layer 7), by running DeepConcolic for ten *feature maps* of each selected layer, and then check neuron coverage and MC/DC for DNNs. For each feature map, the DeepConcolic tool was forced to stop if it did not terminate in 12 hours. Overall, MC/DC requires several hundred thousand test cases and this is in contrast to neuron coverage that only needs a few thousand test cases.

In Figure 2, we give the coverage results with respect to the feature maps. That is, each time DeepConcolic is applied to generate tests for test conditions w.r.t a particular feature map. It seems that feature maps in the later layer are more difficult to cover, and exhibit a higher standard variation in the coverage results. This is likely to be solved by running the tool multiple times.

Figure 3 reports the percentage of adversarial examples in the test cases generated. Test conditions defined by the MC/DC variants for DNNs demonstrate the strength in capturing the adversarial examples and the ability to find a much higher

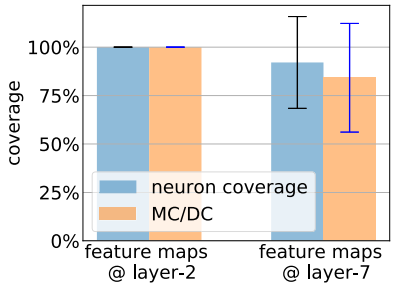


Fig. 2: Coverage results with respect to feature maps

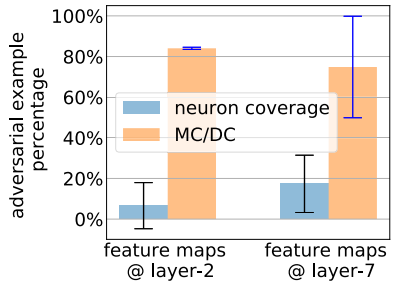
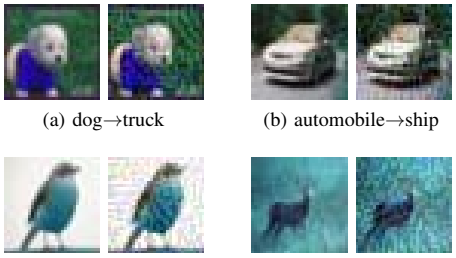


Fig. 3: Adversarial examples detected at each feature map



(a) dog→truck (b) automobile→ship

(c) bird→cat (d) deer→frog

Fig. 4: Selected adversarial examples

level of adversarial examples than neuron coverage. Still, it is interesting to see that, in the case of neuron coverage, more adversarial examples are detected in the deeper layer; this may indicate that neuron activation in later layers has a stronger impact on adversarial behaviour.

Figure 4 exhibits several adversarial examples detected. We also plot the distribution of total adversarial examples from MC/DC for DNNs in Figure 5, which shows that they detect smaller input perturbations in the deeper layer. In a DNN, later layers can be seen as refinements of earlier layer activity, and the test conditions defined by the MC/DC variants for DNNs indeed capture this relationship.

IV. Related Work

Existing works (e.g., [8]) on finding adversarial examples apply heuristic search algorithms based on gradient descent or evolutionary techniques. They are not able to answer what these generated adversarial examples represent or how similar or different they are from each other.

Several coverage guided test case generation methods that may answer these questions have been recently proposed. Neuron coverage [9] measures the percentage of neurons

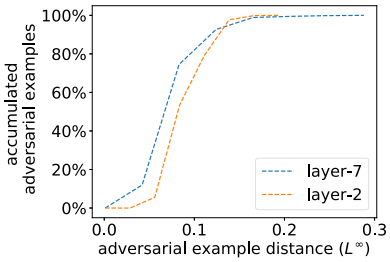


Fig. 5: Accumulated adversarial examples within each distance

activated by the test suite. Its extensions in [10] specify test conditions for a set of neurons (at the same layer). The MC/DC variants for DNNs [5] are the only coverage criteria to capture the causal relation between features from adjacent layers in a DNN. In [12] the approximate nearest neighbours algorithm is used to to guide fuzzing in DNNs. The coverage in [13] requests that when the test dataset is projected into specified finite partitions, there needs to be at least one data point in each region with no less than the associated weight. In [14], [15], the input space is discretised into hyper-rectangles such that one test case is generated for each hyper-rectangle. When the hyper-rectangle is small enough, this approach exhaustively searches the input space. The "Boxing Clever" method in [16] evaluates training data by dividing the inputs space into a series of hyper-rectangles.

The concolic engine of DeepConcolic is related to formal verification of DNNs. Typical symbolic verification solutions (e.g., [17], [18], [19], [20]) based on SAT/SMT or MILP can only scale to DNNs with a few hundred hidden neurons. Recent work [21], [22] using global optimisation is able to handle larger networks. By contrast, concolic testing [6] at each step only symbolically encodes part of the overall DNN's behaviour.

V. Conclusions

DeepConcolic is a tool for testing and debugging deep neural networks. Test case generation is guided by neuron coverage and MC/DC variants for DNNs that are likely to suit different criticality levels. We believe that a worthy application of such a white-box testing tool is to assess the internal structures of a DNN, and that this helps inform safety and robustness arguments for DNNs.

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

References

[1] X. Huang et al., "Safety and trustworthiness of deep neural networks: A survey," *arXiv preprint arXiv:1812.08342*, 2018.

[2] H. Zhu, P. A. Hall, and J. H. May, "Software unit test coverage and adequacy," *ACM Computing Surveys*, vol. 29, no. 4, pp. 366–427, 1997.

[3] K. Hayhurst, D. Veerhusen, J. Chilenski, and L. Rierson, "A practical tutorial on modified condition/decision coverage," NASA, Tech. Rep., 2001.

[4] Y. Sun, M. Brain, D. Kroening, A. Hawthorn, T. Wilson, F. Schanda, F. J. G. Jimenez, S. Daniel, C. Bryan, and I. Broster, "Functional requirements-based automated testing for avionics," in *22nd International Conference on Engineering of Complex Computer Systems*. IEEE, 2017.

[5] Y. Sun, X. Huang, D. Kroening, J. Sharp, M. Hill, and R. Ashmore, "Structural test coverage criteria for deep neural networks," *arXiv preprint arXiv:1803.04792*, 2018.

[6] Y. Sun, M. Wu, W. Ruan, X. Huang, M. Kwiatkowska, and D. Kroening, "Concolic testing for deep neural networks," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE*, 2018, pp. 109–119.

[7] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.

[8] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *International Conference on Learning Representations (ICLR)*, 2014.

[9] 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*. ACM, 2017, pp. 1–18.

[10] L. Ma, F. Juefei-Xu, J. Sun, C. Chen, T. Su, F. Zhang, M. Xue, B. Li, L. Li, Y. Liu, J. Zhao, and Y. Wang, "DeepGauge: Comprehensive and multi-granularity testing criteria for gauging the robustness of deep learning systems," in *Automated Software Engineering (ASE), 33rd IEEE/ACM International Conference on*, 2018.

[11] Y. Tian, K. Pei, S. Jana, and B. Ray, "DeepTest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th international conference on software engineering*. ACM, 2018, pp. 303–314.

[12] A. Odena and I. Goodfellow, "TensorFuzz: Debugging neural networks with coverage-guided fuzzing," *arXiv preprint arXiv:1807.10875*, 2018.

[13] C.-H. Cheng, C.-H. Huang, and H. Yasuoka, "Quantitative projection coverage for testing ML-enabled autonomous systems," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2018.

[14] M. Wicker, X. Huang, and M. Kwiatkowska, "Feature-guided black-box safety testing of deep neural networks," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2018, pp. 408–426.

[15] M. Wu, M. Wicker, W. Ruan, X. Huang, and M. Kwiatkowska, "A game-based approximate verification of deep neural networks with provable guarantees," *arXiv preprint arXiv:1807.03571*, 2018.

[16] R. Ashmore and M. Hill, "Boxing clever: Practical techniques for gaining insights into training data and monitoring distribution shift," in *First International Workshop on Artificial Intelligence Safety Engineering*, 2018.

[17] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *International Conference on Computer Aided Verification*. Springer, 2017, pp. 3–29.

[18] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," in *International Conference on Computer Aided Verification*. Springer, 2017, pp. 97–117.

[19] N. Narodytska, S. P. Kasiviswanathan, L. Ryzhyk, M. Sagiv, and T. Walsh, "Verifying properties of binarized deep neural networks," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[20] M. Akintunde, A. Lomuscio, L. Maganti, and E. Pirovano, "Reachability analysis for neural agent-environment systems," in *Sixteenth International Conference on Principles of Knowledge Representation and Reasoning*, 2018.

[21] W. Ruan, X. Huang, and M. Kwiatkowska, "Reachability analysis of deep neural networks with provable guarantees," in *IJCAI*, 2018, pp. 2651–2659.

[22] W. Ruan, M. Wu, Y. Sun, X. Huang, D. Kroening, and M. Kwiatkowska, "Global robustness evaluation of deep neural networks with provable guarantees for L0 norm," *arXiv preprint arXiv:1804.05805v1*, 2018.