Kexin Pei *Columbia University* **Yinzhi Cao** *The Johns Hopkins University*
Junfeng Yang, Suman Jana *Columbia University*

Editors: Nic Lane and Xia Zhou

DEEPXPLORE: Automated Whitebox Testing of Deep Learning Systems

Excerpted from "DeepXplore: Automated Whitebox Testing of Deep Learning Systems" from ACM SOSP 2017, *Proceedings of the 26th ACM Symposium on Operating Systems Principles* with permission. https://dl.acm.org/citation.cfm?id=3132785

Over the past few years, Deep Learning (DL) has made tremendous progress, achieving or surpassing human-level performance for a diverse set of tasks, including image classification, speech recognition, and playing games like Go. These advances have led to widespread adoption and deployment of DL in security- and safety-critical systems, such as self-driving cars, malware detection, and aircraft collision avoidance systems.

This wide adoption of DL techniques presents new challenges as the predictability and correctness of such systems are of crucial importance. Unfortunately, DL systems, despite their impressive capabilities, often demonstrate unexpected or incorrect behaviors in corner cases for several reasons, such as biased training data, overfitting, and underfitting of the models. In safety- and security-critical settings, such incorrect behaviors can lead to disastrous consequences, such as a fatal collision of a self-driving car. For example, a Google self-driving car recently crashed into a bus because it expected the bus to yield under a set of rare conditions but the bus did not [1]. A Tesla car in autopilot crashed into a trailer because the autopilot system failed to recognize the trailer as an obstacle due to its "white color against a brightly lit sky" and the "high ride height" [2]. Such corner cases were not part of Google's or Tesla's test set and thus never showed up during testing.

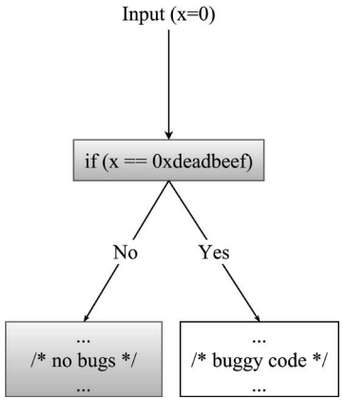
Therefore, safety- and security-critical DL systems, just like traditional software, must be tested systematically for different corner cases to detect and fix ideally any potential flaws or undesired behaviors. This presents a new systems problem as automated and systematic testing of large-scale, real-world DL systems with thousands of neurons and millions of parameters for all corner cases is extremely challenging.

The standard approach for testing DL systems is to gather and manually label as much real-world test data as possible [3, 4]. Some DL systems such as Google self-driving cars also use simulation to generate synthetic training data [5]. However, such simulation is completely unguided as it does not consider the internals of the target DL system. Therefore, for the large input spaces of real-world DL systems (e.g., all possible road conditions for a self-driving car), none of these approaches can hope to cover more than a tiny fraction (if any at all) of all possible corner cases.

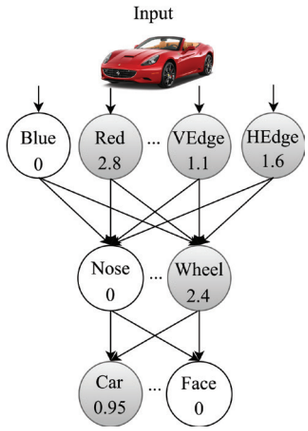
Recent works on adversarial deep learning [6, 7, 8] have demonstrated that carefully crafted synthetic images by adding minimal perturbations to an existing image can fool state-of-the-art DL systems. The key idea is to create synthetic images such that they get classified by DL models differently than the original picture but still look the same to the human eye. While such adversarial images expose some erroneous behaviors of a DL model, the main restriction of such an approach is that it must limit its perturbations to tiny invisible changes or require manual checks. Moreover, just like other forms of existing DL testing, the adversarial images only cover a small part (52.3%) of DL system's logic according to our evaluation. In essence, the current machine learning testing practices for finding incorrect corner cases are analogous to finding bugs in traditional software by using test inputs with low code coverage and thus are unlikely to find many erroneous cases.

The main challenges in automated systematic testing of large-scale DL systems are twofold: (1) how to generate inputs that trigger different parts of a DL system's logic and uncover different types of erroneous behaviors, and (2) how to identify erroneous behaviors of a DL system without manual labeling/checking. We describe how we design and build DeepXplore to address both of these challenges.

Photo, istockphoto.com



(a) A program with a rare branch



(b) A DNN for detecting fars and faces

FIGURE 1. Comparison between program flows of a traditional program and a neural network. The nodes in gray denote the corresponding basic blocks or neurons that participated while processing an input.

OUR SOLUTION TO THE CHALLENGE

Neuron Coverage: To better understand the first challenge, i.e., the problem of low test coverage of rules learned by a DNN, we provide an analogy to a similar problem in testing traditional software. Figure 1 shows a side-by-side comparison of how a traditional program and a DNN handle inputs and produce outputs. Specifically, the figure shows the similarity between traditional software and DNNs: in the software program, each statement performs a certain operation to transform the output of previous statement(s) to the input to the following statement(s), while in DNN, each neuron transforms the output of previous neuron(s) to the input of the following neuron(s). Of course, unlike traditional software, DNNs do not have explicit branches but a neuron's influence on the downstream neurons decreases as the neuron's output value gets lower. A lower output value indicates less influence and vice versa. When the output value of a neuron becomes zero, the neuron does not have any influence on the downstream neurons.

As demonstrated in Figure 1a, the problem of low coverage in testing traditional software is obvious. In this case, the buggy behavior will never be seen unless the test input is 0xdeadbeef. The chances of randomly picking such a value is very small. Similarly, low-coverage test inputs will also leave different behaviors of DNNs unexplored. For example, consider a simplified neural network, as shown in Figure 1b, that takes an image as input and classifies it into two different classes: cars and faces. The text in each neuron (represented as a node) denotes the object or property that the neuron detects, and the number in each neuron is the real value outputted by that neuron. The number indicates how confident the neuron is about its output. Note that randomly picked inputs are highly unlikely to set high output values for the unlikely combination of neurons. Therefore, many incorrect DNN behaviors will remain unexplored even after performing a large number of random tests. For example, if an image causes neurons labeled as "Nose" and "Red" to produce high output values and the DNN misclassifies the input image as a car, such a behavior will never be seen during regular testing as the chances of an image containing a red nose (e.g., a picture of a clown) is very small.

As DL system logic is not expressed in terms of control flow, existing traditional coverage metrics are not applicable to such systems. Therefore, we introduce the concept of neuron coverage for measuring the parts of a DL system's logic exercised by a set of test inputs based on the number of neurons activated (i.e., the output values are higher than a threshold) by the inputs. Particularly, we define neuron coverage of a set of test inputs as the ratio of the number of unique activated neurons for all test inputs and the total number of neurons in the DNN. We consider a neuron to be activated if its output is higher than a threshold value (e.g., 0). Note that this is just one way of defining neuron coverage (analogous to line coverage for traditional code). In the future, we plan to explore other ways of defining neuron coverage based on the sets of different activated neurons.

Differential Testing: To address the second challenge, we leverage multiple DL systems with similar functionality (e.g., self-driving cars by Google, Tesla, and GM) as cross-referencing oracles to identify erroneous corner cases without manual checks. For example, if one self-driving car decides to turn left while others turn right for the same input, one of them is likely to be incorrect. Such differential testing techniques have been applied successfully in the past for detecting logic bugs without manual specifications in a wide variety of traditional software [9]. In this paper, we adapt differential testing for systematically testing DL systems.

DeepXplore: To the best of our knowledge, DeepXplore is the first efficient whitebox testing framework for large-scale DL systems. DeepXplore formulates the problem of

generating test inputs that maximize neuron coverage of a DL system while also exposing as many differential behaviors (i.e., differences between multiple similar DL systems) as a joint optimization problem. Unlike traditional programs, the functions approximated by most popular Deep Neural Networks (DNNs) used by DL systems are differentiable. Therefore, their gradients with respect to inputs can be calculated accurately given whitebox access to the corresponding model. Therefore, DeepXplore uses these gradients to efficiently solve the above-mentioned joint optimization problem for large-scale real-world DL systems. In addition to maximizing neuron coverage and behavioral differences between DL systems, DeepXplore also supports adding custom constraints by the users for simulating different types of realistic inputs (e.g., different types of lighting and occlusion for images/videos).

RESULTS

We evaluated DeepXplore on 15 state-of-the-art DL models trained using five real-world dataset including Udacity self-driving car challenge data, image data from ImageNet and MNIST, Android malware data from Drebin, and PDF malware data from Contagio/VirusTotal. The completed results can be found in the full technical paper[1].

DeepXplore efficiently found thousands of unique incorrect corner case behaviors in all of the tested systems. Table 1 shows the summary of our results. For all of the tested DL models, on average, DeepXplore generated one test input demonstrating incorrect behavior within one second while running on a commodity laptop. The inputs generated by DeepXplore achieved 34.4% and 33.2% higher neuron coverage on average than the same number of randomly picked inputs and adversarial inputs [6, 7, 8] respectively. We further found that the test inputs generated by DeepXplore can be used to retrain the corresponding DL model to improve classification accuracy as well as identify potentially polluted training data. We achieved up to 3% improvement in classification accuracy by retraining a DL model on inputs generated by DeepXplore compared to retraining on the same number of random or adversarial inputs.

TABLE 1. Summary of DeepXplore results

Dataset	Description	DNNs	Original random testing accuracy	Avg. neuron coverage improvement over random	Avg. violations found by DeepXplore (2000 seeds)
MNIST	Handwritten digits	LeNet variants	98.63%	30.5% → 70%	1,289
ImageNet	Images in 1000 categories	VGG16, VGG19, ResNet15	93.91%	1% → 69%	1,980
Driving	Udacity self-driving dataset	Nvidia Dave-2 variants	99.94%	3.2% → 59%	1,839
Contagio/ VirusTotal	PDF malware	Fully connected	96.29%	18% → 70%	1,048
Drebin	Android malware	Fully connected	96.03%	18.5% → 40%	2,000

CONCLUSION

We have provided a brief overview of DeepXplore, demonstrating its capabilities of finding thousands of erroneous behaviors in different state-of-the-art DNNs[2]. The results and discussion presented in the paper remarks the urgent need of systematic whitebox testing of safety-critical DL systems and, more generally, shares the perspectives and insights from both software engineering and machine learning for developing reliable and secure machine learning systems. ■

Kexin Pei is a second-year PhD student at the Department of Computer Science, Columbia University. He is interested in machine learning, and focused on the security and reliability of machine learning systems.

Yinzhi Cao is an assistant professor in computer science at Johns Hopkins University. He earned his PhD in Computer Science at Northwestern University. His research focuses mainly on the security and privacy of the Web, smartphones, and machine learning.

Junfeng Yang is an associate professor in computer science at Columbia University, and co-director of the Software Systems Lab. He received his PhD and his master's degrees in Computer Science from Stanford. His research interests center on building reliable, secure, and fast systems.

Suman Jana is an assistant professor in the department of computer science at Columbia University. His primary research interest is in the field of computer security and privacy. His work has led to reporting and fixing of approximately 250 high-impact security vulnerabilities across a wide range of software. His research software has also been incorporated as part of Google's malware detection infrastructure, Mozilla Firefox, and Apache Cordova.

[1] The full technical paper is at https://arxiv.org/abs/1705.06640.

[2] The source code is available at https://github.com/peikexin9/deepxplore

REFERENCES

[1] Google-accident 2016. "A Google self-driving car caused a crash for the first time." http://www.theverge.com/2016/2/29/11134344/google-self-driving-car-crash-report. (2016).

[2] Tesla-accident 2016. "Understanding the fatal Tesla accident on Autopilot and the NHTSA probe." https://electrek.co/2016/07/01/understanding-fatal-tesla-accident-autopilot-nhtsa-probe/. (2016).

[3] Fei-Fei, Li. "*ImageNet: Crowdsourcing, benchmarking & other cool things.*" CMU VASC Seminar. Vol. 16. 2010.

[4] 2016. *Report on autonomous mode disengagements for waymo self-driving vehicles in california*. https://www.dmv.ca.gov/portal/wcm/connect/42aff875-7ab1-4115-a72a-97f6f24b23cc/Waymofull.pdf?MOD=AJPERES&CVID=

[5] 2017. "Inside Waymo's secret world for training self-driving cars." https://www.theatlantic.com/technology/archive/2017/08/inside-waymos-secret-testing-and-simulation-facilities/537648/. (2017).

[6] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. "Explaining and harnessing adversarial examples." In *Proceedings of the 3rd International Conference on Learning Representations*. https://arxiv.org/abs/1412.6572

[7] Anh Nguyen, Jason Yosinski, and Jeff Clune. 2015. "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images." In *Proceedings of the 28th IEEE Conference on Computer Vision and Pattern Recognition.*

[8] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. "Intriguing properties of neural networks." In *Proceedings of the 2nd International Conference on Learning Representations.*

[9] William M McKeeman. 1998. "Differential testing for software." *Digital Technical Journal* (1998).