A Novel Testing Framework For Vision Models Using Bayesian Network

By

Arooj Arif

aa3506phd

Mini-thesis is submitted for the probation review of PhD

July 2024

Northeastern University London, London - UK

Spring, 2024

Final Approval

This Mni-thesis titled

A Novel Testing Framework For Vision Models Using Bayesian Network

By

Arooj Arif

aa3506phd

has been approved
For the Northeastern University, London

Chair:____
Alex Freitas,
Professor, Department of Computer Science,
University of Kent, Kent

Supervisor:____
Alexandros Koliousis,
Associate Professor, Department of Computer Science,
Northeastern University London, UK

Co-Supervisor:____
Dr.Elena Botoeva,
Lecturer, Department of Computer Science,
University of Kent, Kent

ABSTRACT

Deep learning models, particularly vision models (VMs), are critical in high-stake domains such as autonomous driving, medical diagnostics, and security systems. The real-world deployment of VMs require rigorous robustness testing due to diverse environmental conditions. Current testing approaches primarily focus on neuron coverage. Although this metric is critical; however, it alone does not ensure comprehensive coverage of all corner cases, which can lead to unexpected failures, thus leaving a gap in the overall evaluation of the VMs robustness. My research develops a comprehensive testing framework designed to enhance the evaluation of VMs through a structured five-stage process. The initial stage, Specification Module, focuses on clearly defining all necessary properties of the system to guide the entire testing process and ensure comprehensive coverage. The second stage, Sampling, involves to gather all relevant samples necessary for thorough model testing. In the third stage, Test Case Generation, the properties are specified in the first stage are applied to the collected samples, and test cases are generated accordingly. For example, in autonomous car testing, properties such as dust, noise, rain, and night conditions are considered to evaluate model performance under these conditions. The fourth stage, Testing & Probabilistic Graph, begins with testing the generated test cases to validate their effectiveness. After testing, robustness assessments are conducted both locally and globally. Locally, the robustness of model is evaluated within individual categories or classes to identify weaknesses. In contrast, globally, the model's performance is assessed across various categories to enhance its generalisation capabilities across different scenarios. Errors are systematically recorded for later analysis. This stage integrates a probabilistic approach using Bayesian network, combined with solid mathematical formulation, to provide a comprehensive visual and quantitative analysis of the model's performance at both local and global levels. The final stage, Error Summarization, compiles and analyses the recorded errors, producing actionable graphical error reports and recommendations for VMs refinement.

TABLE OF CONTENTS

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Overview

Deep Neural Networks, or DNNs, are increasingly being used in diverse applications owing to their ability to match or exceed human level performance. The availability of large datasets, fast computing methods and their ability to achieve good performance has paved way for DNNs into safety-critical avenues such as autonomous car driving, medical diagnosis, security, etc. The safety-critical nature of such applications makes it imperative to adequately test these DNNs before deployment. However, unlike traditional software, DNNs do not have a clear control-flow structure. They learn their decision policy through training on a large dataset, adjusting parameters gradually using several methods to achieve desired accuracy. Consequently, traditional software testing methods like functional coverage, branch coverage, etc. cannot be applied to DNNs, thereby challenging their use for safety critical applications. A lot of recent work, discussed in chapter II, has looked into developing testing frameworks for DNNs. These methods suffer from certain limitations, as discussed in . In our work, we intend to make an effort to overcome these limitations and build a fast, scalable, efficient, generalizable testing framework for deep neural networks.

In this section of thesis, the background and motivation, Research Questions, contributions and organization of thesis have been presented.

1.1.1 Background and motivation

In the past few years, deep neural networks (DNNs) have made remarkable progress in achieving human-level performance. With the broader deployment of DNNs on various safety critical systems like autonomous, healthcare, avionics, etc., the concerns over their safety and trustworthiness have been raised in public, particularly highlighted by incidents involving self-driving cars.

An important low-level requirement for DNNs is that they are robustness against input perturbations. DNNs have been shown to suffer from a lack of robustness because of their susceptibility to adversarial examples such that small modifications to an input, sometimes imperceptible to humans, can make the network unstable.

In this thesis, we examine existing testing methods for deep neural networks, the opportunities for improvement and the need for a fast, scalable, generalizable end-to-end testing method.

Coverage criteria for traditional software programs, such as code coverage and branch coverage check that all parts of the logic in the program have been tested by at least one test input and all conditions have been tested to independently affect the entailing decisions. On similar lines, any coverage criterion for deep neural networks must be able to guarantee completeness, that is, it must be able to ensure that all parts of the internal decision-making structure of the DNN have been exercised by at least one test input.

Generating or selecting test inputs in a guided manner usually has two major goals - maximizing the number of uncovered faults, and maximizing the coverage.

Testing DNNs for correctness involves verifying behaviors against a ground truth or oracle. The traditional approach, collecting and manually labeling real-world data, is labor-intensive. Another method compares outputs across multiple DNNs for the same task, identifying discrepancies as corner cases. However, this can misclassify inputs if all models agree, due to shared biases or errors. This comparative approach is further limited to tasks with multiple reliable models, which may not always be available, especially in innovative or specialized applications.

1.1.2 Challenges of Deep Learning Models

The growing use of deep neural networks in safety critical applications makes it necessary to carry out adequate testing to detect and correct any incorrect behavior for corner case inputs before they can be actually used. Deep neural networks lack an explicit control-flow structure, making it impossible to apply to them traditional software testing criteria such as code coverage

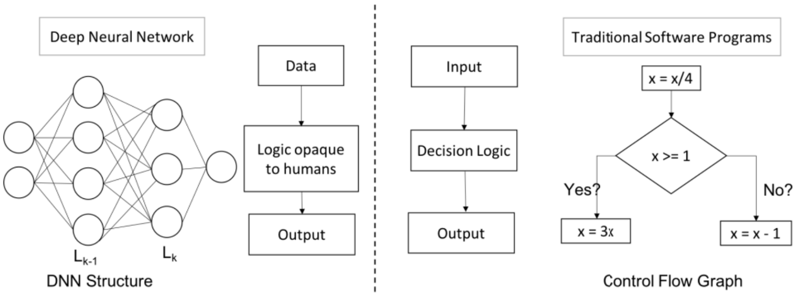

FIGURE 1.1: The internal logic of a deep neural network is opaque to humans, as opposed to the well laid out decision logic of traditional software programs [1]

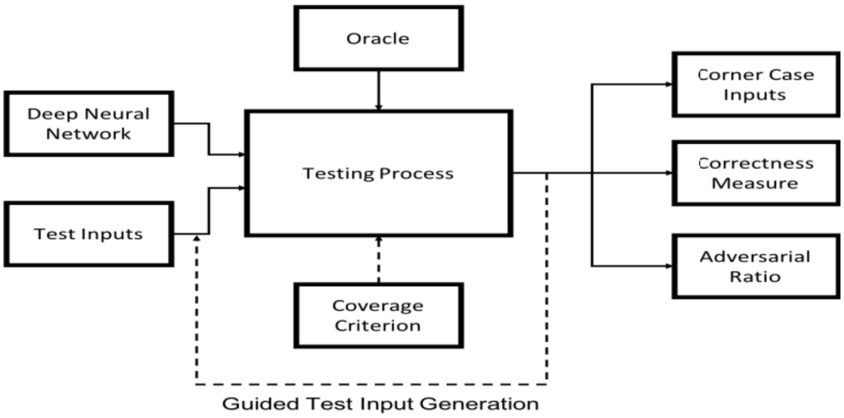

FIGURE 1.2: A high-level representation of most existing DNN testing methods [1]

- input space is extremely large, **unguided simulation** are higly unlikely to find erroneous behavior

1.1.3 Challenges in Testing of Deep Learning Models

unlike traditional software, DNNs do not have a clear control-flow structure. They learn their decision policy through training on a large dataset, adjusting parameters gradually using several methods to achieve desired accuracy. Consequently, traditional software testing methods like functional coverage, branch coverage, etc. cannot be applied to DNNs, thereby challenging their use for safety- critical applications.Traditional software testing methods fail when applied to DNNs because the code for deep neural networks holds no information about the internal decision-making logic of a DNN.

DNNs testing techniques aim to discover bugs, through finding counter examples that challenge the system's correctness, or to establish confidence by rigorously evaluating the system with numerous test cases. These testing techniques are computationally less expensive and therefore are able to work with state-of-the-art DNNs. However, DL testing has some limitations:

- Standards available in industry but **Lack of Logcial Structure** and **System Specification**
- Heavily depend on manual collections of test data under different conditions which become expensive as number of test condition increases
- existing coverage criteria are not detailed enough to notice subtle behaviours exhibited by DL systems.

coarse coverage criteria, open ended processes, unreliable oracles, inefficient test input generation methods, inability to scale to larger DNNs and different network architectures

1.1.4 Research Questions

- How to specify relevant local robustness properties?
- How we sample inputs efficiently?
- How to design comprehensive framework?

1.1.5 Thesis Contributions

The idea is to propose coverage criteria to evaluate the adequacy of test inputs. A good test input should be surprising enough to challenge the system, revealing behaviors not seen during training, but not so surprising that it becomes irrelevant or unrealistic compared to the training data.

a test case generation method is proposed to generate test inputs that have greater coverage and which uncover greater corner case behavior.

- abc

1.1.6 Organization of thesis

The remainder of the thesis is organized as follows: related studies are presented in Chapter **??**. System model and proposed methodology are demonstrated in Chapter **??**. Chapter **??** describes the simulation results of our proposed schemes.

Finally, the findings of this work along with future directions are presented in Chapter 5.

Chapter 2

Literature

2.1 Overview

2.1.1 Coverage Criteria for Deep Learning Models

Existing Coverage Methods	**Description**	**Limitation**
Neuron Coverage	Measures the model's logic use by counting activated neurons from test inputs.	Doesn't capture all potential DNN behaviors and can achieve high coverage with few inputs; it is a coarse measure.
k-Multisection Neuron Coverage	Divides neuron activation values observed during training into k buckets and counts how many buckets are covered by a set of inputs.	Loses information on activations beyond the observed range during aggregation.
DeepCover	Considers condition-decision dependencies between adjacent DNN layers.	Limited to small, feedforward, fully-connected networks; doesn't generalize to complex architectures like RNNs or LSTMs.
DeepCT	Inspired by combinatorial testing, assesses logic use by the fraction of neurons activated in each layer.	Lacks consideration for inter-layer relationships and hasn't been proven to scale to real-world DNNs.

Table 2.1: Coverage Methods, Descriptions, and Limitations

2.1.2 Test Case Generation for Deep Learning Models

Existing Methods	**Description**	**Limitation**
Joint Optimization	Modifies an existing input through image manipulations recursively until it triggers different behavior in the model.	Time-consuming and produces a low ratio of impactful test inputs compared to the total tested/generated.
Greedy Search	Applies random transformations to an existing test input until a suitable test input is identified.	Similar to joint optimization, it is also time-intensive and results in a low number of effective test inputs relative to the total tested.

Table 2.2: Summary of Existing Test Input Generation Methods

Existing Approaches	**Limitations**
Collecting as much real-world data as possible and manually labeling it for correctness.	The process requires a lot of manual effort.
Comparing outputs across multiple DNNs for the same task, identifying discrepancies as corner cases.	Can misclassify inputs if all models agree, due to shared biases or errors. Limited to tasks with multiple reliable models, which may not always be available.

Table 2.3: Limitations of Existing Approaches in DNN Testing

Table 2.4: Summary of Test Methodologies and Their Characteristics

Methodology	**Dataset**	**Benchmark**	**Limitation/Future Work**	**Coverage Criteria**	**Test Generation**
Symbolic execution with local explainability. LIME provides explanations for predictions[2]	German Credit Data, Adult census income, Bank marketing, US Executions, Fraud Detection, Raw Car Rentals, Credit data, Census data	THEMIS (Algorithm) The technique produces 3.72 times more successful test cases than existing state-of-the-art.	FW: Expand to text and image domains FW: Measure symbolic execution efficacy using neuron coverage, boundary value coverage.	–	Symbolic/Concolic
Concolic testing method [3]	- MNIST - CIFAR-10	DeepXplore, DeepTest, DeepCover, and DeepGauge		NC, SSC, NBC	Concolic

Table 2.4: Summary of Test Methodologies and Their Characteristics

Methodology	**Dataset**	**Benchmark**	**Limitation/Future Work**	**Coverage Criteria**	**Test Generation**
Whitebox framework for testing DL systems, introducing neuron coverage for test measurement [4]	- MNIST - ImageNet - Driving - VirusTotal - Drebin	LeNet variations State-of-the-art image classifiers Nvidia DAVE PDF malware detectors Android app malware detectors	Inherits differential testing constraints.	NC	Dual-optimisation
Automates testing for DNN-driven autonomous cars [5]	Udacity self driving car challenge 2	Chauffeur- Epoch Rambo-S1 Rambo-S2 Rambo-S3	L:missing some realistic cases. L: restricted only steering angle, not focus on brake and accelerator L: cannot simulate complex road scene .	NC	Greedy search

Table 2.4: Summary of Test Methodologies and Their Characteristics

Methodology	**Dataset**	**Benchmark**	**Limitation/Future Work**	**Coverage Criteria**	**Test Generation**
White box methodology, Proposed four novel test criteria tailored to DNN, structural features. able to capture and quantify causal relations existing in a DNN, Achieved balance between bug finding ability and computational cost [6]	MNIST, CIFAR-10, ImageNet	State-of-the-art neural networks of different sizes (from a few hundred up to millions of neurons) to demonstrate their utility with respect to four aspects: bug finding, DNN safety statistics, testing efficiency, DNN internal structure analysis		MC/DC	Symbolic execution

Table 2.4: Summary of Test Methodologies and Their Characteristics

Methodology	**Dataset**	**Benchmark**	**Limitation/Future Work**	**Coverage Criteria**	**Test Generation**
Proposed criteria facilitate the understanding of DNNs as well as the test data quality from different levels and angles[7]	MNIST, ImageNet	LeNet-1 LeNet-4 LeNet-5, VGG-19, ResNet-50	More diverse datasets and DL architectures needed. Check on real-world systems.	NBC	Gradient descent methods
An unsupervised learning framework to synthesize realistic driving scenes to test inconsistent behaviors[8]	Udacity Training Udacity Test Ep1 Udacity Test Ep2 Youtube Ep1 Youtube Ep2	Autumn, Chauffeur	Lack a good standard to evaluate image quality (i.e., realism). Udacity dataset is relatively small and the autonomous driving models are quite simple. Only focus on steering wheel.	–	Mutation testing

Table 2.4: Summary of Test Methodologies and Their Characteristics

Methodology	**Dataset**	**Benchmark**	**Limitation/Future Work**	**Coverage Criteria**	**Test Generation**
An automated fuzz testing framework for hunting potential defects of general-purpose DNNs[9]	MNIST, CIFAR-10, ImageNet	LeNet-1 LeNet-4 LeNet-5 RN-20 VGG-16 MobileNet RN-50	NC cannot generate effective results to evaluate the models with various quality. NC is less effective in error triggering test detection and sensitive defect detection.	NC KMNC NBC SNAC KNC KNC	Metamorphic mutation

Chapter 3

Proposed Framework

Proposed Approach

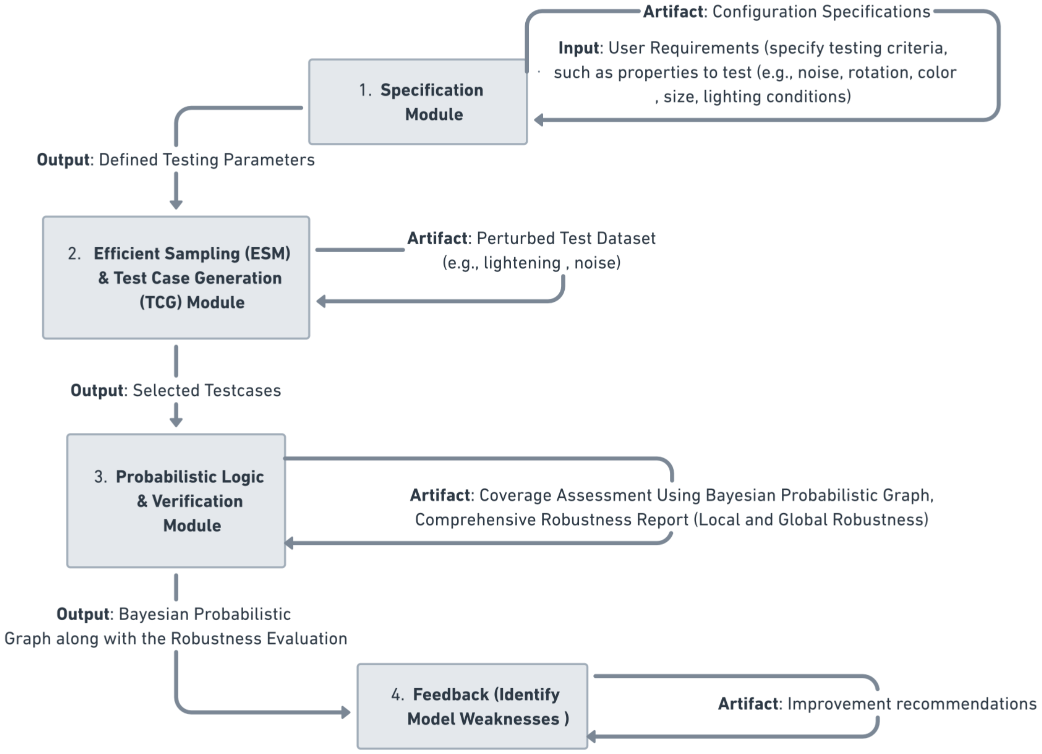
Artifact: Configuration Specifications
Input: User Requirements (specify testing criteria, such as properties to test (e.g., noise, rotation, color , size, lighting conditions)
1. Specification Module
Output: Defined Testing Parameters
2. Efficient Sampling (ESM) & Test Case Generation (TCG) Module
Artifact: Perturbed Test Dataset (e.g., lightening , noise)
Output: Selected Testcases
3. Probabilistic Logic & Verification Module
Artifact: Coverage Assessment Using Bayesian Probabilistic Graph, Comprehensive Robustness Report (Local and Global Robustness)
Output: Bayesian Probabilistic Graph along with the Robustness Evaluation
4. Feedback (Identify Model Weaknesses)
Artifact: Improvement recommendations

FIGURE 3.1: The outline of proposed approach explains overall flow of work.

3.0.1 Bayesian Network-based Coverage Metrics

Two testing coverage metrics are defined in Figure.3.2 : the local coverage (LC) and the global coverage (GC).

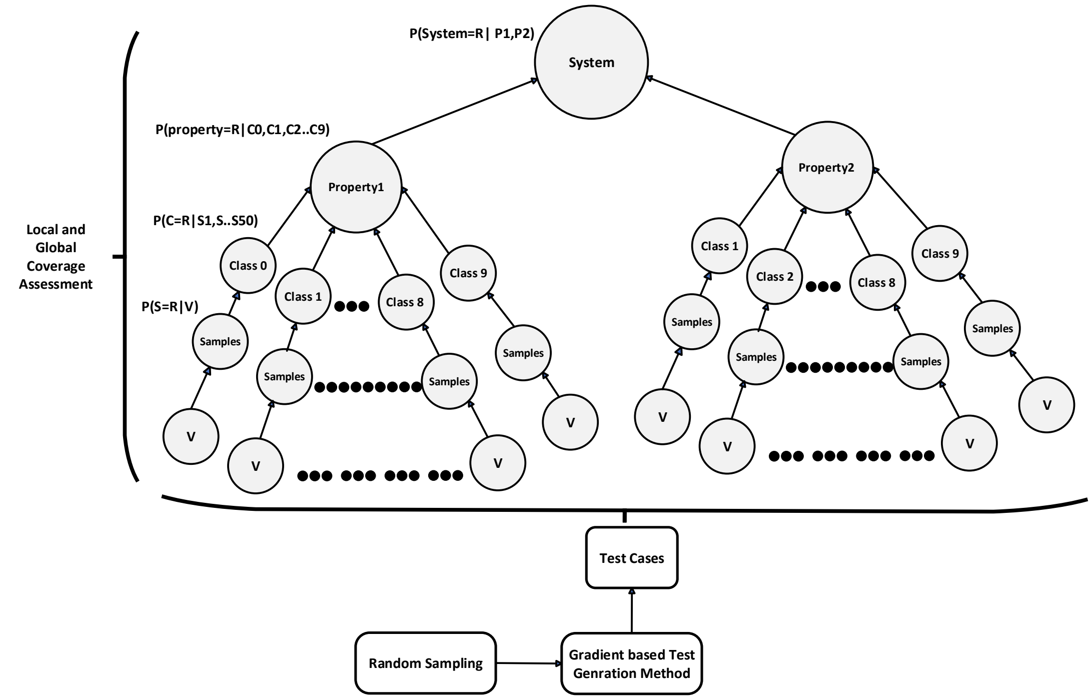

Figure 3.2: Coverage Assesment (Local and Global)

3.0.2 Gradient based Test Generation

Algorithm 1: Test Case Generation via Gradient-Based Attacks

```
Input:      Model $\mathcal{M}$ with bounds [0, 1],
            Set of images $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$,
            Corresponding labels $\mathcal{L} = \{l_1, l_2, \ldots, l_n\}$,
            Perturbation magnitudes $\mathcal{E} = \{\epsilon_1, \epsilon_2, \ldots, \epsilon_k\}$,
            Set of attacks $\mathcal{A} = \{A_1, A_2, \ldots, A_m\}$.
Output:     Set of test cases $\bigcup TestCases_{ij}$.
Procedure:  GenerateTestCases($\mathcal{M}$, $\mathcal{I}$, $\mathcal{L}$, $\mathcal{E}$, $\mathcal{A}$)
              for each attack $A_j$ in $\mathcal{A}$
                for each $\epsilon_i$ in $\mathcal{E}$
                  Generate testcases $Adv_{ij} = A_j(\mathcal{M}, \mathcal{I}, \epsilon_i)$
                  Verify the $Adv_{ij}$ to obtain $V_{ij}$
                  Evaluate $V_{ij}$ against $\mathcal{L}$ to determine $isRobust_{ij}$
                  Compile test cases $TestCases_{ij} = \{Adv_{ij}, isRobust_{ij}\}$
                end for
              end for
              return $\bigcup TestCases_{ij}$
```

Chapter 4

Simulations and Results

Chapter 5

Conclusion and future work

5.1 Conclusion

In this thesis, we have analyzed issues related to security, computational overhead and conditional privacy in existing vehicular ad dissemination schemes. We have proposed a BC-based ad dissemination scheme that enables conditional anonymity by using pseudonyms instead of ZKPoK, as well as efficient proof verification by using batch verification. The proposed scheme reduces computational cost and enables malicious vehicle detection. Also the vehicle tracing attacks are mitigated by updating the pseudonyms of vehicles after a regular interval. Furthermore, a BC based reputation management system is proposed to promote efficient reputation sharing in VENs. The proposed system initiates with the registration of vehicles through the usage of real and pseudo identities. The registration is done via CA, which also maps the $PIDs$ of the vehicles with their $RIDs$. The $PIDs$ are generated with the help of ECDSA, which ensures conditional anonymity and traceability. In the underlying system, distributed revocation is ensured via SSS algorithm. Moreover, the storage overhead is reduced using IPFS. The reputation data is stored in IPFS while the hashes generated by IPFS are stored in BC. Simulations are performed and the results show the efficacy of the proposed system in terms of computational cost and storage overhead. 18-20% reduction in computational overhead and 35-40% reduction in storage overhead are observed when using the proposed system. In the end, the security analysis on the bases of replay attack, 51% attack and smart contract vulnerabilities prove the model's robustness.

5.2 Future work

In future, we will further improve the performance of our proposed scheme by employing Cuckoo Filters and IOTA Tangle DLT. Moreover, we will develop a mechanism to reuse the pseudonyms that are previously revoked in order to reduce the storage overhead on RSUs.

Chapter 6

References

References

[1] Sekhon, Jasmine, and Cody Fleming. ”Towards improved testing for deep learning.” 2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER). IEEE, 2019.

[2] Agarwal, Aniya, et al. ”Automated test generation to detect individual discrimination in AI models.” arXiv preprint arXiv:1809.03260 (2018).

[3] Sun, Youcheng, et al. ”Concolic testing for deep neural networks.” Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. 2018.

[4] Pei, Kexin, et al. ”DeepXplore.” Communications of the ACM 62.11 (2019): 137-145.

[5] Tian, Yuchi, et al. ”Deeptest: Automated testing of deep-neural-network-driven autonomous cars.” Proceedings of the 40th international conference on software engineering. 2018.

[6] Sun, Youcheng, et al. ”Testing deep neural networks.” arXiv preprint arXiv:1803.04792 (2018).

[7] Ma, Lei, et al. ”Deepgauge: Multi-granularity testing criteria for deep learning systems.” Proceedings of the 33rd ACM/IEEE international conference on automated software engineering. 2018.

[8] Zhang, Mengshi, et al. ”DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems.” Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. 2018.

[9] Xie, Xiaofei, et al. ”Deephunter: Hunting deep neural network defects via coverage-guided fuzzing.” arXiv preprint arXiv:1809.01266 (2018).

[10] Gopinath, Divya, et al. ”Symbolic execution for deep neural networks.” arXiv preprint arXiv:1807.10439 (2018).