DeepEvolution: A Search-Based Testing Approach for Deep Neural Networks

Houssem Ben Braiek and Foutse Khomh

SWAT Lab., Polytechnique Montréal, Montréal, Canada

{*houssem.ben-braiek, foutse.khomh*}*@polymtl.ca*

Abstract— **The increasing inclusion of Deep Learning (DL) models in safety-critical systems such as autonomous vehicles have led to the development of multiple model-based DL testing techniques. One common denominator of these testing techniques is the automated generation of test cases, e.g., new inputs transformed from the original training data with the aim to optimize some test adequacy criteria. So far, the effectiveness of these approaches has been hindered by their reliance on random fuzzing or transformations that do not always produce test cases with a good diversity. To overcome these limitations, we propose, DeepEvolution, a novel search-based approach for testing DL models that relies on metaheuristics to ensure a maximum diversity in generated test cases. We assess the effectiveness of DeepEvolution in testing computer-vision DL models and found that it significantly increases the neuronal coverage of generated test cases. Moreover, using DeepEvolution, we could successfully find several corner-case behaviors. Finally, DeepEvolution outperformed Tensorfuzz (a coverage-guided fuzzing tool developed at Google Brain) in detecting latent defects introduced during the quantization of the models. These results suggest that search-based approaches can help build effective testing tools for DL systems.**

I. INTRODUCTION

Deep Neural Networks (DNN)-based software systems are considered to be the next generation of software [1], thanks to their innovative development paradigm, where the program logic is inferred automatically from data using statistical learning methods. Recently, they have been deployed in large-scale and critical systems such as self-driving cars. However, ensuring the quality assurance of DNN-based software is still very challenging as evidenced by recent deadly incidents with Uber's cars[1]. In fact, because of their non-deterministic nature and the absence of a reference oracle, it is very challenging to reason about the behavior of a DNN and hence to test it. Novel testing techniques are needed both during model engineering and deployment phases, to guarantee the reliability and robustness of in-production DNN-based software. During the model engineering phase, developers need to assess the impact of their configuration choices carefully. The effectiveness of this assessment depends on the capability of testing data to trigger both the major functionalities of the model (regular cases) and the minor functionalities (corner cases). When deploying a trained DNN model in an embedded system or a cell phone, a quantization [2] of the model is often required to fit into this constrained environments (i.e., limited storage

and computation resources). A post-deployment testing phase is required to assess the effect of this quantization on the reliability and the robustness of the model. In fact, the change in precision that occur during quantization increases the likelihood of coincidental correctness in the long sequences of linear and non-linear operations performed by DNNs. Therefore, the challenge is to generate testing inputs that are resilient to this phenomenon and, hence, capable of checking for the existence of inconsistencies and unexpected behaviors in the quantized model.

In this paper, we propose DeepEvolution, a novel Search-based Software Testing (SBST) approach for DNNs models. DeepEvolution aims to detect inconsistencies and potential defects in DNN models. DeepEvolution relies on population-based metaheuristics to explore the search space of semantically-preserving metamorphic transformations. Using a coverage-based fitness function to guide the exploration process; it aims to ensure a maximum diversity in the generated test cases. We assessed the effectiveness of DeepEvolution on popular image recognition DNNs, i.e., LeNet [3] and CifarNet [4]. Results show that DeepEvolution succeeds in boosting the neuronal coverage of DNNs under test, finding multiple erroneous DNN behaviors. DeepEvolution also outperformed Tensorfuzz [5] in detecting latent defects introduced during quantization.

The remainder of this paper is organised as follows.
Section II introduces the software testing concepts adapted by our approach. Section III presents the testing flow of DeepEvolution. Section IV describes an instantiation of DeepEvolution to test computer-vision DNNs. Section V reports evaluation results, while Section VI discusses threats to their validity. Section VII summarizes the most relevant related works and Section VIII concludes the paper.

II. BACKGROUND ON SOFTWARE TESTING

In the following, we briefly describe the fundamental software testing techniques that have been used and adapted by our proposed approach for DL testing.

A. Metamorphic Testing

Metamorphic testing [6] is a pseudo-oracle testing technique that allows finding erroneous behaviors by detecting violations of identified metamorphic relations (MR). The first step of this technique is the construction of MRs that relate inputs in a way that the relationship between

---

[1] https://www.nytimes.com/2018/05/07/technology/uber-driverless-car-crash.html

their corresponding outputs becomes known in prior, so the desired outputs for generated test cases can be expected. For example, a metamorphic relation for testing the implementation of the function $\sin(x)$ can be the transformation of input x to $\pi - x$ that allows examining the results by checking if $\sin(x)$ differs from $\sin(\pi - x)$. Using MRs, a partial oracle can be generated automatically to test the program. Program's inputs are transformed following the MRs and expected outputs are computed. Any significant difference between the expected output and the output produced by the program under test indicates a defect in the program.

B. Code Coverage Criteria

Test adequacy evaluation consists of assessing the fault-revealing ability of existing test cases. It is based on different adequacy criteria that estimate if the generated test cases are 'adequate' enough to terminate the testing process with confidence that the program under test is implemented properly. Code Coverage is one of the most used test adequacy evaluation criteria. It gauges the proportion of the programs source code that is executed by test cases. In fact, test cases achieving high code coverage are more likely to uncover more defects, since they trigger more code execution paths.

C. Search-Based Testing

The generation of test inputs with the aim of achieving high coverage is a hard problem that random testing often fails to solve (because of the size and complexity of software under test). Search-based software testing (SBST) techniques have been introduced to overcome the limitations of random testing. SBST techniques formulate the test coverage criteria as a fitness function that compares and contrasts candidate solutions in terms of their ability to cover new program's states. Using this fitness function, SBST techniques leverage metaheuristics, i.e., gradient-free optimizers requiring only few or no assumptions on the fitness function and inputs data, to drive the search into a promising area of the input space in order to generate effective test cases that help reaching an acceptable level of coverage in reasonable time.

III. DEEPEVOLUTION: TESTING WORKFLOW

DeepEvolution aims to generate effective synthetic test cases from an existing test data. Instead of searching in the space of inputs of a model, DeepEvolution explores the space of transformations looking for interesting transformations that are able to provide effective test cases. Using a population-based metaheuristics, it iteratively evolves an initial set of candidate transformations, deriving new transformations that satisfy the following criteria: (i) they are significantly different from their parents to produce test data exhibiting new DNN's behaviors, and (ii) achieve high fitness values (to keep the search in relevant discovered regions). From one generation of candidates to another, DeepEvolution performs follow-up tests with the resulting transformed inputs and stores the failed tests that exhibit erroneous DNNs behavior or induce a divergence between the original DNN and its quantized version. The evolution process stops when a predefined number of generations is attained. DeepEvolution requires as input: (i) a set of metamorphic transformations; (ii) a coverage-based fitness function capable of comparing different transformations based on the effectiveness of their produced test inputs; and (iii) a population-based metaheuristic. The fitness function should capture both local (neurons covered by a mutated input that were not covered by its corresponding original input) and global (neurons covered by a mutated input that were not covered by all previous test inputs) neuronal coverage. In the following we present an instantiation of DeepEvolution for computer vision.

IV. DEEPEVOLUTION: COMPUTER-VISION INSTANCE

Given the rapid progress and the impressive performance of DNNs in computer-vision tasks [7], we propose following the instantiation of DeepEvolution components for testing computer-vision DNN models.

A. Metamorphic Transformation

First, we gather a list of parametric image-based transformations that can be organised in two groups:

1) **Pixel-value transformations:** change image contrast, image brightness, image blur, image sharpness and random perturbations within a limited interval.
2) **Affine transformations:** image translation, image scaling, image shearing, and image rotation.

Because each image-based transformation has a theoretical domain, which defines the interval of possible values of its parameters such as brightness factor or rotation angle θ, when applying transformations, we need to take into account these domain boundaries to ensure that transformed inputs are semantically equivalent to the original ones. To infer the valid domain interval of each transformation's parameters, we manually tune them to set up the appropriate range of values, i.e., high and low boundaries, that preserves the input semantics before and after each transformation, with respect to the data distribution.

To enable a large-scale generation of synthetic inputs from existing labeled testing data, we build a compound metamorphic transformation that assembles all the image-based transformations described above, in order to enhance the changeability of mutations and the diversity of generated inputs. Its application on a given image consists of applying the supported pixel-value transformations in sequence, and then, performing each single affine transformation once, on the resulting mutated input. We opted for this conservative strategy that consists of applying only one affine transformation following the pixel-value transformations because applying multiple affine transformations at once would increase the chances of generating meaningless images, i.e., images that don't occur in real-world situations.

Our defined metamorphic transformation produces the following mutated inputs: inputs resulting from only pixel-value transformations and inputs that are the results of applying, separately, each one of the affine transformations. To verify that generated inputs remain semantically equivalent to the original inputs, we compute a Structural Similarity Index

(SSIM) [8] which assesses the similarity between two images based on the visual impact of three characteristics: luminance, contrast, and structure. We expect that pixel-based mutated inputs differs from their originals with respect to these characteristics, but a very low SSIM indicates that the new image looses mostly all the information inherited from its parent. Therefore, we reject mutated synthetic inputs for which SSIM values are lower than a tuned threshold.

B. DNN Coverage

We adapt the Neuron Coverage (NC) metric proposed by Pei et al. [9] to capture two levels of coverage (i.e., local and global) for each test input.

Local neurons coverage (NLNC): this represents the new neurons covered by the mutated test input that have not been covered by its corresponding original test input.

Global neurons coverage (NGNC): this consists of the new neurons covered by the mutated test input that have not been covered by all the previous test inputs, including both genuine and synthetic test inputs.

We define the following fitness function:

$$Fitness = \alpha \times NLNC + \beta \times NGNC \qquad (1)$$

α and β are weights assigned to each coverage measure.

C. Swarm-based Metaheuristics

We encode our compound metamorphic transformations as a vector, where each component represents one parameter that may be related to either a pixel-value or an affine transformation. To ensure semantically preserving transformations, we use the valid domain intervals of transformations that we have already tuned manually to create the high and low boundaries vectors, defining the subspace of exploration. We instantiate DeepEvolution using the following 7 swarm-based metaheuristics. (1) Particle Swarm Optimization (PSO) [10], (2) Cuckoo Search Algorithm (CSA), (3) Bat Algorithm (BAT) [11], (4) Gray Wolf Optimizer (GWO) [12], (5) Moth Flame Optimizer (MFO) [13], (6) Whale Optimization Algorithm (WOA) [14], (7) Multi-Verse Optimizer (MVO) [15]. These metaheuristics algorithms are flexible enough to be easily applicable to a broad class of constrained optimization problems involving high dimensional bounded real-valued vectors without any prior search space discretization.

V. EMPIRICAL EVALUATION

We assess the effectiveness of DeepEvolution through the following three research questions:

RQ1: How much can DeepEvolution increase the coverage of generated test cases?

RQ2: Can DeepEvolution detect diverse erroneous behaviors in DNN models?

RQ3: Can DeepEvolution detect divergences induced by DNN quantization?

A. Experiment Setup

Datasets. We selected the two popular publicly available datasets, MNIST [16] and CIFAR-10 [17], as our evaluation subjects. Since neuronal coverage estimations and models post-execution analysis are computation-intensive tasks, we decided to take random samples from each of our studied test datasets as initial testing data. More specifically, we randomly selected two samples D_1 and D_2 from each dataset; with increasing size (i.e., respectively 50 and 100).

DNNs. For each dataset (i.e., MNIST and CIFAR-10), we took, respectively, the official open-source implementation of Tensorflow models, LeNet [3] and CifarNet [4] to allow the reproducibility of our results and comparisons with our approach.

Settings of DeepEvolution. We adopt the default configurations of metaheuristics (which is a conservative approach) and we choose $\alpha = 0.1$ and $\beta = 1.0$ for the fitness function, which is consistent with their corresponding measure magnitude and our priority of increasing the neuron coverage. We select the common hyper-parameters, $population size = 10$ and $max iterations = 10$. To avoid the effects of randomness, all results are averaged over 3 runs or more.

B. RQ1: DNN Coverage Increase

Motivation. We aim to evaluate if the generated test data can help increasing the neuronal coverage, i.e., triggering neurons, which are not covered by the original test data.

Findings. DeepEvolution significantly boosts the neuronal coverage. Table I shows the final neuronal coverage ratio achieved by each implemented swarm-based metaheuristic. The results show that the test data generated by all the studied metaheuristic algorithms significantly enhance the two coverage measures, as confirmed by the Wilcoxon Signed Rank tests. Although the obtained neuronal coverage

TABLE I
THE REACHED NEURON COVERAGE PER METAHEURISTIC

Meta heuristic	MNIST		CIFAR-10	
	D_1	D_2	D_1	D_2
Traditional	44.77	50.89	48.03	53.16
BAT	94.85	96.35	96.02	97.99
CS	94.74	96.12	96.78	98.30
GWO	92.77	94.55	96.32	97.83
MFO	93.11	95.10	95.64	97.52
MVO	86.57	90.06	93.76	96.54
PSO	91.11	94.04	95.50	97.49
WOA	94.55	95.91	95.85	97.68

measures were generally high, the searching process reaches almost a stationary value when it could no longer improve the global coverage induced by the generated inputs, and as a consequence, its role becomes limited to only finding transformed inputs pushing the DNN to behave differently. Nevertheless, the augmentation of the original test data lend a refreshing boost that enabled the enhancement of neuronal coverage, which shows that adding more original instances enlarges the inputs search space to cover more possible test cases. This suggests that the quality of the initial input data is important for successfully covering the major patterns learned by the DNN model under test and increasing the chances of producing rare test inputs.

We noticed that *MVO* performs slightly worse than the others. This can be explained by its tendency to exploit more around the best candidates previously found to converge quickly and not to go further in exploring solutions far from the best recognized regions. This characteristic is however helpful when a metaheuristic optimizer is used to find an optimal global solution at the end, but since our objective is to explore the maximum of relevant regions in the space, we need to increase the exploration ability of *MVO*, we can fix its starting parameters that emphasize the exploration such as higher TDR (i.e., the distance of maximum variation around the best solution) and lower WEP (i.e., probability of generating new candidates around the best solution).

Furthermore, similarly to the usage of test coverage in traditional software testing, increasing the neuronal coverage has been shown to be an effective way to enhance the diversity of the generated inputs; allowing uncovering rare corner-case behaviors, and potentially, intensifying their defect-revealing ability. The effectiveness of our search-based approach in detecting defects is the main purpose of the next two research questions.

C. RQ2: Detection of DNN Erroneous Behaviors

Motivation. The objective is to assess the effectiveness of our approach in testing the robustness of the DNN; by finding misclassified synthetic inputs.
Findings. DeepEvolution can effectively generate test cases that trigger erroneous behaviors of the DNN. Table II presents erroneous behaviors detected by each metaheuristic algorithm. As all metaheuristic algorithms succeeds to

TABLE II

NUMBER OF ERRONEOUS BEHAVIORS PER METAHEURISTIC

Meta heuristic	MNIST		CIFAR-10	
	D_1	D_2	D_1	D_2
BAT	488	963	317	642
CS	1567	3499	1533	3001
GWO	1298	2411	1046	1929
MFO	1343	2955	1098	2387
MVO	378	774	370	764
PSO	1116	2913	1108	2403
WOA	1702	3601	1122	2335

reveal defects of the studied DNNs, it indicates that generating synthetic test inputs towards improving the neuronal coverage could trigger more states of a DNN, incurring higher chances of defect detection, which is consistent with the practical purpose of testing criteria used in traditional software testing. Indeed, the augmentation of sample data size, from D_1 to D_2 has significantly increased the number of erroneous behaviors detected. We obtained almost the double by doubling the input data size. This result suggests that DeepEvolution is capable of obtaining adversarial inputs for each original input and that the local coverage level integrated in the fitness function plays an important role in assessing how much the DNN's state of the transformed input is different from the state that resulted from the original input. Thus, it is capable of finding corner-cases testing inputs even if the global neuronal coverage reaches higher values; as evidenced by the increase in the triggered erroneous behaviors when augmenting the initial test data

despite no significant improvement in the coverage between the two dataset samples. The results of *MVO* reinforce the previous observation about their lack of exploration capability. The default implementation of *BAT* also exhibits a similar insufficiency of diversification that could be remedied using adaptive rates of pulse emission r (i.e., the probability of adjusting the found solutions) and loudness A (i.e., the probability of generating a new candidate randomly).

D. RQ3: DNN Quantization Defects

Motivation. The goal is to assess the usefulness of Deep-Evolution in finding difference-inducing inputs that expose potential quantization defects.
TensorFuzz [5] performs a coverage-guided fuzzing process to generate mutated inputs that are able to expose disagreements between a DNN trained on MNIST (that is 32-bit floating point precision) and its quantized versions where all weights are truncated to 16-bit floating points. We use it as baseline to assess DeepEvolution. To ensure a fair comparison, we fix the configuration of TensorFuzz, including the data corpus size and number of mutations per element, in a way that the two approaches (TensorFuzz and DeepEvolution) produces the same number of test cases from each original test input.
Findings. DeepEvolution can effectively detect defects introduced during DNN quantization, outperforming the coverage-guided fuzzing tool TensorFuzz. Table III presents the number of synthetic test data that were able to induce a difference between the DNN's outcomes (difference-inducing inputs); exposing quantization defects. As can be seen, all the implemented metaheuristics suc-

TABLE III

NUMBER OF QUANTIZATION DEFECTS PER METAHEURISTIC

Test Method	D_1	D_2
TensorFuzz	8	17
BAT	32	70
CS	71	136
GWO	26	103
MFO	39	78
MVO	29	66
PSO	42	86
WOA	24	69

ceeded in exposing quantization defects and most of them outperformed TensorFuzz in terms of number of difference-inducing inputs found, confirming our intuition (and the motivation behind DeepEvolution) that by enabling the optimisation of coverage criteria, metaheuristic-based searching techniques can help increase diversity in generated test cases and hence improve their efficiency.

VI. THREATS TO VALIDITY

In this section, we discuss potential threats to the validity of our work and highlight our mitigation measures.

The selection of experimental subjects (i.e., dataset and DNN models) is a threat to the generalizability of our results. We mitigate this threat by using practical model sizes and commonly-studied MNIST and CIFAR-10 datasets. For each studied dataset, we choose to use official TF implementation with their corresponding configuration in order to avoid

possible implementation bugs or misunderstanding issues that could hinder our evaluation process. Another threat could be the choice of parameters; we selected equal values for the two hyper-parameters, $populationsize = 10$ and $maxiterations = 10$ because some metaheuristic algorithms rely on the iterative evolution of population and others rely on the availability of multiple candidates, so we choose the same value for them to make the evaluation as fair as possible. The selection of metaheuristic algorithms could be a threat to validity. We choose to implement swarm-based metaheuristics because of their randomness and non-deterministic nature that allowed them to be effective in resolving huge space problems. Furthermore, we implement several algorithms, because the No Free Lunch Theorem (NFL) states that no algorithm could outperform all other algorithms with regard to all optimization problems. However, when evaluating DeepEvolution with different metaheuristics, we do not compare their performance in solving our testing objective, since we use their default configuration and do not perform any hyperparameters tuning. In fact, we expect their performance to increase if they are tuned to fit our optimisation problem. The manual tuning of metamorphic transformations' domain and the threshold of SSIM could affect the validity of our results. To mitigate this threat, we selected a sample of our generated images using a confidence level of 95% and an error margin of 5%, and verified them manually. We found them to be correct.

VII. RELATED WORK

Pei et al. [9] proposed DeepXplore, the first white-box testing framework specialized for DNN, which has two main components: (i) a new coverage metric specialized for DNNs (named neuron coverage) that estimates the amount of activated neurons and (ii) a differential testing component that uses multiple DNNs' implementations to solve the same problem (cross-referencing oracles in order to circumvent the lack of a reference oracle). Building on DeepXplore, Guo et al. proposed DLFuzz [18], where mutation was restricted to the imperceptible pixel-value perturbations. Later, Tian et al. proposed DeepTest [19], a tool for automated testing of DNN-driven autonomous cars. In DeepTest, Tian et al. focus on generating realistic synthetic images by applying realistic image transformations like changing brightness, contrast, blurring, and fog effect to mimic different real-world phenomena. Based on the neuron coverage proposed by Pei et al., DeepTest performs a coverage-guided greedy search to finding realistic image transformations that can increase neuron coverage in a self-driving car DNNs. Odena and Goodfellow proposed a coverage-guided fuzzing framework named TensorFuzz [5]. This framework follows the same strategy of transforming new inputs from the original test data in way that maximize discovering novel DNN's states. However, it is based on a simple fuzzing process that consists in continuously adding random noises to inputs that previously triggered uncovered regions of the DNNs, with the hope of uncovering new states. We compared the performance of our proposed DeepEvolution to that of TensorFuzz an found it to be more effective at detecting latent defects introduced during the quantization of DNNs.

VIII. CONCLUSION

This paper presents DeepEvolution, a search-based DL testing approach that leverages semantically-preserving metamorphic transformations, DNN coverage criteria, and population-based metaheuristics. Our evaluation on computer-vision DNNs shows that DeepEvolution can improve the coverage of DNNs and successfully expose corner-cases behaviors. It also outperforms Tensorfuzz in detecting latent defects introduced during the quantization of models.

REFERENCES

[1] A. Karpathy, "Software 2.0," *https://medium.com/@karpathy/software-2-0-a64152b37c35*, 2018.

[2] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[3] "Variant of lenet," https://github.com/tensorflow/models/blob/master/research/slim/nets/l accessed: 2019-04-02.

[4] "Cifar10," https://github.com/tensorflow/models/blob/master/research/slim/nets/cifarnet. accessed: 2019-04-02.

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

[6] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: a new approach for generating next test cases," Technical Report HKUST-CS98-01, Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong, Tech. Rep., 1998.

[7] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, "Deep learning for computer vision: A brief review," *Computational intelligence and neuroscience*, vol. 2018, 2018.

[8] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.

[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] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on*. IEEE, 1995, pp. 39–43.

[11] X.-S. Yang, "A new metaheuristic bat-inspired algorithm," in *Nature inspired cooperative strategies for optimization (NICSO 2010)*. Springer, 2010, pp. 65–74.

[12] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Advances in engineering software*, vol. 69, pp. 46–61, 2014.

[13] S. Mirjalili, "Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm," *Knowledge-Based Systems*, vol. 89, pp. 228–249, 2015.

[14] S. Mirjalili and A. Lewis, "The whale optimization algorithm," *Advances in Engineering Software*, vol. 95, pp. 51–67, 2016.

[15] S. Mirjalili, S. M. Mirjalili, and A. Hatamlou, "Multi-verse optimizer: a nature-inspired algorithm for global optimization," *Neural Computing and Applications*, vol. 27, no. 2, pp. 495–513, 2016.

[16] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[17] A. Krizhevsky, V. Nair, and G. Hinton, "The cifar-10 dataset," *http://www.cs.toronto.edu/kriz/cifar.html*, 2014.

[18] J. Guo, Y. Jiang, Y. Zhao, Q. Chen, and J. Sun, "Dlfuzz: differential fuzzing testing of deep learning systems," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2018, pp. 739–743.

[19] 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.