A Combinatorial Approach to Testing Deep Neural Network-based Autonomous Driving Systems

Jaganmohan Chandrasekaran
Department of Computer Science & Engineering
The University of Texas at Arlington
Arlington, USA
jaganmohan.chandrasekaran@mavs.uta.edu

Yu Lei
Department of Computer Science & Engineering
The University of Texas at Arlington
Arlington, USA
ylei@cse.uta.edu

Raghu Kacker, D. Richard Kuhn
Information Technology Lab
National Institute of Standards and Technology
Gaithersburg, USA
{raghu.kacker, d.kuhn}@nist.gov

***Abstract*—Recent advancements in the field of deep learning have enabled its application in Autonomous Driving Systems (ADS). A Deep Neural Network (DNN) model is often used to perform tasks such as pedestrian detection, object detection, and steering control in ADS. Unfortunately, DNN models could exhibit incorrect or unexpected behavior in real-world scenarios. There is a need to rigorously test these models with real-world driving scenarios so that safety-critical bugs can be detected before their deployment in the real world.**

In this paper, we propose a combinatorial approach to testing DNN models. Our approach generates test images by applying a set of combinations of some basic image transformation operations to a seed image. First, we identify a set of valid transformation operations or simply transformations. Next, we design an input parameter model based on the valid transformations and generate a t-way (t=2) combinatorial test set. Each test represents a combination of transformations, and can be used to produce a test image. We execute the test images on a DNN model and distinguish between consistent and inconsistent behavior using a relation. We conducted an experimental evaluation of our approach on three DNN models that are used in the Udacity challenge. Our results suggest that test images generated by our approach can effectively identify inconsistent behaviors and can significantly increase neuron coverage. To the best of our knowledge, our work is the first effort to use a combinatorial testing approach to generating test images based on image transformations for testing DNNs used in ADS.

Keywords—Testing DNN models, Combinatorial Testing, Deep Learning Testing, Neural Network Testing, Testing Self-driving cars, Testing autonomous vehicles

I. INTRODUCTION

Recent years have seen significant advancements in the field of deep learning. For traditional software applications, a developer explicitly writes the programming logic based on a specific set of requirements. In contrast, deep learning software applications use a deep neural network (DNN) to derive its decision logic from a training dataset, which typically includes a large number of data instances. Deep learning applications have exhibited an extraordinary ability to discover valuable insights and derive complex decision logic from the training dataset. They have been used to perform tasks, such as image recognition, object detection, and language translation, with a high degree of precision.

Deep learning has been applied in many application domains that are considered to require human intelligence. In particular, deep learning plays a significant role in the operation of autonomous driving systems, where DNN models are used to perform tasks such as obstacle detection, pedestrian detection, steering control, perception and localization, and route planning. However, since DNN models are trained and evaluated using a training dataset, they may suffer from the generalizability problem. For example, an investigation into Uber's accident suggests that their driving software system failed to consider the scenario of jaywalking pedestrians [38]. Tesla's autopilot failed to distinguish between a bright sky and a white trailer crossing an intersection; the autopilot attempted to drive under the trailer resulting in a loss of life [34]. Accidents reported in [34, 38] suggest a critical need to rigorously test DNN models, especially using tests that imitate the real-world conditions and include corner-case scenarios.

Recent work suggests that synthetic images generated using image transformation techniques can effectively identify the inconsistent behavior of DNN models [36, 44, 46]. Zhang et al. proposed a framework that uses a Generative Adversarial Network (GAN) based unsupervised technique to generate synthetic images that mimic the two extreme weather conditions (snow and rain) [46]. The findings from their study suggest that the DNN models used in the Udacity driving challenge exhibit several inconsistencies when executed with test inputs generated using their approach. Tian et al. demonstrated that testing the DNN model with synthetic images generated with basic image transformations can produce inconsistent behavior [36]. Their results suggest that synthetic images generated by combining different image transformations increase neuron coverage, a measure of proportion of neurons activated in a DNN model.

This paper presents a combinatorial testing-based approach to generating test images to test DNN models. In our approach, we first identify a set of basic image transformations that do not change the ground truth of the image being transformed. That is, in principle, the prediction result for a transformed image produced by such a transformation is the same as the original image. (In practice, the prediction result of the transformed image may be different from that of the original image by a small amount that is less than a certain threshold.) We then use Combinatorial Testing (CT) to generate a t-way test set that covers every t-way combination of these transformations. Each test is a combination of transformations and can be used to create a test image.

To address the test oracle problem, we consider how to identify inconsistent behaviors in two cases. In the first case, the ground truth of a test image remains the same as that of the original image. Thus, we consider that an inconsistent behavior is detected if the prediction result of a test image differs from that of the original image by an amount that is more than a threshold. In the second case, the ground truth of a test image may be different from that of the original image. Thus, the prediction result of a test image may be expected to be very different from that of the original image. In this case, we compare the prediction results of the same test image from different DNNs that perform the same prediction. An

inconsistent behavior is detected if the prediction results of a test image from different models do not agree with each other.

Our approach's novelty lies in the fact that we generate test images using CT. The key insight behind CT is that while the behavior of a system could be affected by many factors, individual failures are typically caused by a very small number of factors [21]. We hypothesize that this insight also applies to testing DNNs. That is, inconsistent behaviors of a DNN model could be triggered by a combination of a small number of basic image transformations. In another word, a t-way test set that covers every t-way combination of image transformation can be effective to detect inconsistent behaviors.

We report an experimental evaluation of our approach using three of the top five models, namely Autumn [4], Chauffeur [7], and Rambo [29], from the Udacity self-driving challenge. We generate tests by applying t-way transformations to the seed images selected from the Udacity test dataset. Our results show that t-way tests can identify a number of inconsistent behaviors in these DNN models. For example, out of 121 t-way tests generated for a seed image, 29 tests and 95 tests resulted in an inconsistent behavior for the Autumn model and Chauffeur model, respectively. Our results suggest that a small number of tests (121 tests) can significantly increase the cumulative neuron coverage compared to its baseline. In some cases, t-way tests covered more than ten times of additional neurons compared to their respective baseline. Overall, the results provide initial support for our hypothesis. The results indicate that t-way tests can help the practitioners to effectively test DNN models in terms of both detecting inconsistent behavior and increasing neuron coverage.

Combinatorial testing is applied to test DNN models, as reported in [9, 23]. However, they follow a white box testing approach by testing the neurons' interactions within each layer in the DNN [23] and the effect of variable strength-based CT tests on interactions between pre-layer and post-layer neurons of the DNN [9]. To the best of our knowledge, we believe the work reported in this paper is the first effort to apply the combinatorial testing approach to generate test images by combining different types of image transformations to test DNN models used in autonomous driving systems.

The remainder of this paper is organized as follows. In Section II, we provide a brief introduction to DNN based software systems and combinatorial testing. In Section III, we present our approach, in terms of the major steps performed in the testing process. In Section IV, we report an experimental evaluation, where we first report the design of the evaluation and then discuss the experimental results. Section V discusses the existing work that is related to ours. Section VI provides concluding remarks and directions for our future work.

II. Background

A. DNN based software systems

Deep learning is a machine learning technique that uses DNN to perform tasks such as classification and regression. In traditional software systems, a developer derives rules from the requirements and implements the rules in the form of program logic. In contrast, DNN based software systems derive their decision logic from an input dataset; the decision logic is referred to as a trained DNN model. The DNN model takes an input (either image or text depending on the domain) and produces an output in the form of a prediction.

In recent years, DNN models are widely adopted across different domains such as medical imaging, language translation, and autonomous driving systems. They are increasingly deployed in safety-critical fields to perform tasks such as speech recognition, image classification, natural language processing. In particular, autonomous driving systems (ADS) use DNN models to perform tasks such as lane control, object identification, and pedestrian detection. For example, a DNN model used in the autonomous driving system takes an image from the camera as its input and predicts the steering angle.

Based on the application domain, the practitioners use different types of DNN architectures to build a DNN model. Convolutional Neural Network (CNN), a type of neural network architecture, is widely used in the autonomous driving system as they exhibit a higher success rate (better accuracy) in image recognition. Recurrent Neural Network (RNN), a type of neural network architecture that uses temporal information to make predictions, is used in the autonomous driving system to predict steering angles based on a sequence of input data (temporal information). The subject models used in our experiments use a CNN to extract features from the input images that are passed to either an RNN or a fully connected network (FC-network) to predict the steering angle.

B. Combinatorial Testing

Combinatorial Testing is a black-box test generation technique. For a given system under test (SUT), combinatorial testing focuses on systematically testing the interactions among the system's different parameters with a smaller number of tests.

Consider a program P with four parameters and each parameter having three values. To test program P, we will require 81 tests (3*3*3*3=81) to test all possible combinations (exhaustive test set). Compared to this, using a t-way combinatorial test set (t=2), it is possible to test all possible interactions between any two parameters (at least once) with nine tests. In general, combinatorial testing approach can significantly reduce the number of tests [10].

ACTS, a combinatorial test generation tool, uses the IPOG algorithm to generate t-way tests. Consider a program P modeled with an input parameter model (IPM) with k parameters. For any t parameters (out of k) of P, IPOG algorithm generates a t-way test set to cover the first t parameters and then it generates additional tests (i.e., extending the test set) to cover the first t+1 parameters in an iterative manner until all the parameters are covered by the test set [43]. ACTS can generate t-way tests of strength t=2 through t=6.

III. Approach

In this section, we present a combinatorial approach to test DNNs. Figure 1 presents an overview of our approach. The proposed approach is applicable for DNNs that take an image as an input and outputs a prediction. The goal of our approach is to generate synthetic images to test the pre-trained DNN model.

In the first step, we identify basic image transformations that can be used to create synthetic test images. Geometric

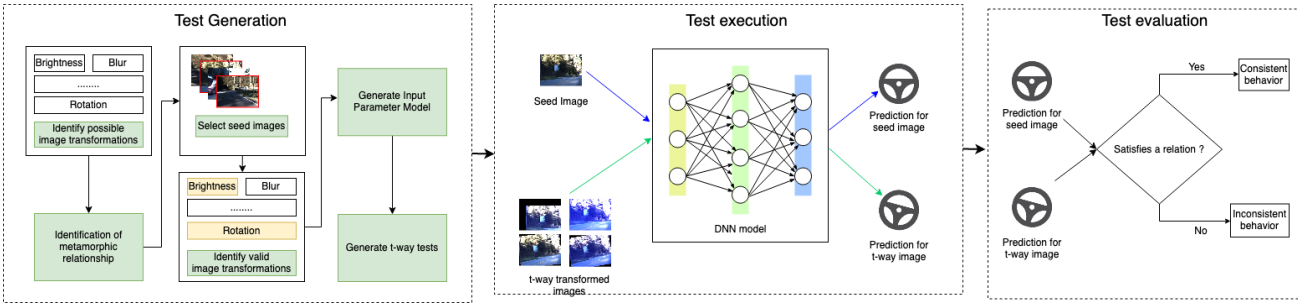

FIGURE 1 – APPROACH OVERVIEW

image transformation techniques such as linear transformations and affine transformations can be used to generate synthetic images. Applying a linear transformation to an image does not change the size or shape of the input image. In contrast, applying the affine transformation, the origins of the transformed image and the original image do not necessarily match with each other. In other words, applying the affine transformation shall result in a change of orientation and size of the original image. We represent a transformation in a two-tuple form *(transformation name, transformation value)*. For example, (Brightness, 10) increases an image's brightness by a value of 10.

In the second step, given the nature of the domain, the test input space for a DNN model can be too large (nearly infinite). Hence, we apply equivalence partitioning, and randomly select a seed image from each partition. For example, a test dataset (used in Autonomous Driving Systems domain) contains image frames recorded all around the year. In this case, we partition the test dataset based on the weather conditions such as sunny, rainy, fog, snow, overcast, and normal weather. We then randomly select an image (seed image) from each group.

Our approach is aimed to generate valid test images (synthetic images). In the first step, we identified a set of transformations that could be applied to generate synthetic images. However, every transformation might not be uniformly applicable to seed images. In other words, given the type and nature of the seed image, applying certain transformations (identified from Step 1) might generate a synthetic image that is either unrealistic or invalid. For example, consider two seed images: image #1 captured during the middle of the day (brighter, sunny day) and image #2 captured around the time of sunset during winter. We generate a synthetic image by applying a transformation -- decrease the brightness by 80% to both seed images. The resulting synthetic image for image #1 could be valid in terms that the image is viewable to human eyes. In contrast, the synthetic image for image # 2 might be invalid, since it could be a completely dark image. As this example illustrates, some image transformations when applied to a seed image might generate invalid test inputs. Note that, in this case, we consider image #2 as invalid. However, in real world, it is still important to test such scenarios using other approaches that deal with images that are completely dark.

To alleviate this problem, in the third step, for each seed image, we identify a set of valid transformations (a subset of all possible transformations). We determine the validity by comparing the prediction results of the original image (Po) and the transformed image (Ps). For DNN models that outputs a continuous value (for example, a steering angle), the transformed image's prediction result may be different, within a degree of tolerance, from that of the original image. Therefore, a transformation is considered valid if the difference between Po and Ps is less than a certain threshold $|Po - Ps| \leq$ threshold.

Consider the following example. We are testing a pre-trained DNN model that predicts the steering angle. Contrast and Rotation are two possible image transformations, with five different values per transformation: (Contrast,1), (Contrast,2), (Contrast, 3), (Contrast, 4), (Contrast, 5), and (Rotation, 2°), (Rotation, 4°), (Rotation, 6°), (Rotation, 8°), (Rotation, 10°). Thus, there exists a total of ten possible transformations. We apply these transformations to the seed image and generate ten synthetic images. Then, the predicted value of a synthetic image is compared with the predicted value of the original image. Three transformations – (Contrast, 5) and (Rotation, 8°), (Rotation, 10°) exceed the threshold. The remaining seven transformations are identified as valid transformations for the seed image. It is often the case that different seed images can have different sets of valid transformations. The motivation behind this step is to generate valid tests, thus minimizing false positives.

In the fourth step, we generate t-way tests. For each seed image, we design an Input Parameter Model (IPM), where each transformation is identified as a parameter, and the transformation values that make a transformation valid are identified as parameter values. In our earlier example, Contrast: {1,2,3,4} and Rotation: {2°, 4°, 6°} are identified as parameters and values.

In the final step, based on the IPM, we generate abstract t-way test set. Then, we derive concrete tests (synthetic images) by applying t-way image transformations to the seed images using the OpenCV framework [49]. The synthetic images are used to test the DNN models.

One challenge in testing ML models is lack of a test oracle. In practice, data labeling is considered to be an expensive and challenging task. Thus, a tester might not be able to determine the ground truth of a synthetic image. In this case, the practitioners can compare the prediction value across different model implementations and identify the inconsistent behavior. Doing so can help practitioners assess a model's performance in the absence of ground truth. In our approach, we evaluate the t-way test results in the following two cases:

Case 1: The original seed image and t-way synthetic image share the same ground-truth value. In this case, for each model, if a test fails to satisfy the relation: $|Po - Ps| \leq$ threshold, the test is considered to exhibit inconsistent behavior.

Case 2: The original seed image and t-way synthetic image do not share the ground-truth value, i.e., they might have a different ground-truth value. In this case, we evaluate a test by comparing its prediction results across multiple models (Pm). It can be challenging to derive the ground truth for each test (synthetic image). Therefore, we define an inconsistent behavior as follows: A test exhibits an inconsistent behavior if the maximum difference in prediction change across multiple models exceeds a threshold value i.e., if a test fails to satisfy the following relation

$$|\max(\mathrm{Pm}) - \min(\mathrm{Pm})| \leq \text{threshold} \quad (1)$$

IV. Experiments

In this section, we present an experimental evaluation of our approach. The source code, data and/or artifacts have been made available at [31, 35]

A. Research Questions

Our experiments are designed to answer the following two research questions:

- Can our combinatorial testing-based approach successfully identify inconsistencies among DNN model implementations?
- How does the combinatorial testing-based approach impact the neuron coverage?

B. Models

We use open-source DNN models from the Udacity self-driving car challenge. Teams participating in the Udacity self-driving car challenge developed DNN models that predict the steering angle (output) based on an image frame (input). Submitted models were evaluated with the Udacity test dataset [30] and ranked based on their prediction accuracy (performance). Models from the Udacity self-driving car challenge are among the widely used subject models to evaluate test generation techniques for testing autonomous vehicle software systems [36][44][46][15].

Among the top five ranking models from the challenge that are publicly available at [40], we select three models, namely Chauffeur [7], Rambo [29], and Autumn [4] as our subject models. We did not use the other two models, namely, komanda [40] and rwightman [40]. For komanda, the pre-trained model weight file is not accessible [20]. For rwightman, the publicly available script failed to execute [30].

- The Autumn model consists of three 5x5 convolution layers with stride 2, followed by two 3x3 convolution layers and five fully connected layers with a dropout [4]. The Autumn model is implemented using Tensorflow(v0.11) and Keras(v1.1.0) [1, 18].
- The Chauffeur model uses a Convolutional Neural Network (CNN) to extract features from the input image and use a Long Short-Term Memory (LSTM) network, a type of Recurrent Neural Network (RNN), to predict the steering angles. Chauffeur model is implemented using Tensorflow (v1.12) and Keras (v1.2.2) [1, 18].
- The Rambo models consist of three CNNs to extract features, and their output is merged in the final layer to predict the steering angle. The Rambo model is implemented using Tensorflow (v1.12) and Theano (v0.9) [1, 33].

For each subject model, the sequence of image frames that has been processed before the current frame impacts the prediction of the current frame. In Autumn, the prediction is based on five consecutive frames (input + four previous frames). Chauffeur's prediction is determined by 100 consecutive frames (input + ninety-nine previous frames). Rambo considers three consecutive frames to make the prediction (input + two previous frames).

We present the model details in Table I. The first and second column list the model name, and its network architecture. The third column presents the Root Mean Square Error (RMSE) value. RMSE is one of the widely used metric to measure the prediction errors of a machine learning model that outputs a continuous value. A lower RMSE value indicates better performance (prediction). All submitted models in the Udacity challenge were evaluated and ranked per RMSE value. In the last column, we present information about the number of sequence images that influence the current frame's prediction.

Model Name	***Architecture Information***	***RMSE***		***Prediction Logic***
		Reported RMSE	***Our RMSE***	
Autumn	CNN	0.04	0.04	Previous 4 frames + current frame
Chauffeur	CNN + RNN	0.06	0.06	Previous 99 frames + current frame
Rambo	CNN	0.06	0.06	Previous 2 frames + current frame

Table I – Model Information

C. Seed Images

We select the seed images from the Udacity test dataset. The test dataset consists of 5614 test images and their respective steering angles [39]. The steering angle is in the range -25° to +25° and normalized to ±1° [36]. An image with a positive steering value indicates the vehicle is turning right. A negative steering value indicates turning left, while a steering angle of 0° or closer to 0° indicates the vehicle is traveling in a straight direction (i.e., no turns).

The steering angle is in the range of -1 to +1. Based on the steering angle, we divide the test images into different groups with an interval of 0.1 per group. We have a total of 20 groups starting from ($-1.0 <$ steering angle ≤ -0.9) through ($0.9 <$ steering angle ≤ 1.0). We refer to these groups by Group 1 through Group 20, respectively.

The test dataset does not contain images in the range (-1.0, -0.9). Thus, there is no representative image from Group 1 in our experiments. For the remaining nineteen groups, we randomly select one image from each group as our seed image. In total, we have nineteen seed images in our experiments.

D. Test Oracle

In the autonomous driving domain, it is hard to determine an exact steering angle for transformed images. Zhang et al. used a method to identify the DNN model's consistent behavior, and it is defined as follows: Given a transformed image as input, if the DNN model predicts (steering angle) within a certain error bound, the model is considered to exhibit a consistent behavior [46]. Similar to their work, we use the

following relation: |Po – Ps| ≤ threshold and identify a model's inconsistent behavior in two cases. Po denotes the steering angle of the original image and Ps denotes the steering angle of the transformed image. The threshold value is a configurable parameter, and we use the following three threshold values: 0.1, 0.2 and 0.3 in our experiments.

In the first case, we assume the t-way synthetic image and the original image shares the ground truth. Thus, a t-way synthetic image that violates this relation |Po – Ps| ≤ threshold exhibits an inconsistent behavior.

In the second case, we assume the synthetic image and the original image does not share the ground truth. In this case, we compare the prediction results of the same synthetic image from three DNNs that perform the same prediction. A t-way synthetic image exhibits an inconsistent behavior if it violates the relation (1).

E. Metrics

We measure our approach's effectiveness by computing the number of inconsistent behaviors identified by a t-way test set. The more inconsistent behaviors the t-way test detects, the more effective the t-way test is considered.

We also use neuron coverage to measure the effectiveness of our approach. The notion of neuron coverage is defined as the ratio of unique neurons that is activated for a given input to the total number of neurons in a DNN [27]. A neuron is considered activated if its output is greater than a certain threshold (defined by the user). Tian et al. used neuron coverage in their experiments and made their artifacts publicly accessible [3, 36]. We use their neuron coverage framework and threshold (0.2) in our experiments. To measure the cumulative neuron coverage, we first load the seed image to the DNN and measure its neuron coverage. This coverage information is used as the baseline in our experiment. Then, we execute the t-way images and calculate cumulative coverage relative to the baseline.

F. Test Generation

We begin the test generation step by identifying the possible image transformations applicable to the Udacity test dataset. Tian et al. applied a set of seven different types of simple image transformations to the Udacity test dataset and studied their impact on neuron coverage [36]. We use these seven image transformations. Table II presents the list of transformations and their values used in our experiments. Overall, we have seventy image transformations (7 different types of transformations * 10 values per transformation).

Recall that, as discussed in section III, every possible transformation might not be uniformly applicable to all the seed images. Therefore, in the next step, we identify the set of valid transformations for each seed image.

1) Identification of valid transformations

We apply the seven types of transformations with ten different values per transformation and generate 70 synthetic images per seed image. Next, the seed image is loaded to three subject models, and their respective predicted steering angle is recorded (Po).

Then, for each model, the synthetic images are loaded as input. Their predicted steering angle (Ps) is compared with the steering angle of the original seed image (Po). A transformation is considered to be valid if |Po – Ps| ≤ 0.1. At the end of this step, we identify the set of valid transformations per seed image.

Transformations		***Values***
Blur	Averaging	3x3, 4x4, 5x5, 6x6
	Gaussian	3x3, 5x5, 7x7
	Median	3, 5
	Bilateral	(9, 75, 75)
Brightness		10, 20, 30, 40, 50, 60, 70, 80, 90, 100
Contrast		1.2, 1.4, 1.6, 1.8, 2.0, 2.2., 2.4, 2.6, 2.8, 3.0
Rotation		3, 6, 9, 12, 15, 18, 21, 24, 27, 30
Scale		1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0
Shear (Horizontal)		0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0
Translation		(10,10), (20,20), (30,30), (40,40), (50,50), (60,60), (70,70), (80,80), (90,90), (100,100)

TABLE II – TRANSFORMATIONS AND VALUES

2) Generation of t-way tests

To create t-way tests, first, we create an input parameter model (IPM). The seven transformations from Table II are identified as parameters. Based on the valid transformations, we identify the set of possible values for each parameter. Next, based on the IPM, we generate abstract t-way tests using the ACTS tool [2]. In our experiments, we generate a 2-way combinatorial test set. Each test represents a combination of transformations that could be applied to the seed image. In the final step, we convert the abstract t-way tests to concrete tests, i.e., generate synthetic images based on t-way tests. Recall that each abstract test represents a combination of transformations. When we generate synthetic images, we apply image transformations in the following order – *Blur, Brightness, Contrast, Rotation, Scale, Shear and Translation* using the OpenCV framework [49]. (We tried to apply the transformations in different orders, and found that the order has minimal impact on the prediction outcomes.) We execute the subject DNN models with t-way concrete tests (i.e., synthetic images) and compare their output with the original image's predicted steering angle.

G. Example

We illustrate our approach with an example. For the seed image from group 2 (*1479425660620933516.jpg*), the three models, namely Chauffeur, Rambo, and Autumn, predict a steering angle -0.760681748390198, -0.62006545, and -0.83253384, respectively (Po).

In Step 1, we generate 70 synthetic images for *1479425660620933516.jpg* based on the transformations listed in Table II. Then, we execute the 70 synthetic images on three models and compare their predicted steering values (Ps) with their respective steering angle prediction of the original image (Po).

In the case of the Chauffeur model, a transformation (synthetic image) is considered valid if |Po – Ps| ≤ 0.1; 48 out of 70 transformations satisfy the criteria and thus considered valid transformations for the chauffeur model. For the Rambo model, a transformation (synthetic image) is considered valid if |Po – Ps| ≤ 0.1; 57 out of 70 transformations satisfy the criteria. Likewise, for the Autumn model, a transformation is considered valid if the absolute value of |Po – Ps| ≤ 0.1, and 33 transformations satisfy the criteria. Among the 70

transformations, 28 transformations are valid across all three models, and hence these 28 transformations are used to generate t-way tests.

In step 2, using the ACTS tool, we create the input parameter model with valid parameters and values identified from the previous step. Then, we generate 121 abstract tests from a 2-way test set. Next, we use the Open-CV framework [49] to generate concrete tests (2-way synthetic images). Finally, we test the subject models using concrete tests.

H. Results and discussion

First, we present the results of synthetic images generated using the transformations and values from Table II. These results are used to identify valid transformations that are later used in the generation of t-way tests. Next, we present the inconsistent behavior detection results of the t-way tests. Finally, we discuss the neuron coverage achieved by the t-way tests.

1) Identification of Valid Transformations

Figure 2 presents the details of valid transformations identified for each group. The x-axis presents the group details and the y-axis presents the number of possible transformations. In our experiments, we have 70 possible transformations (7 transformations * 10 values per transformations). Due to limited space, we present the number of valid transformations per group. Our results suggest that *Group 10* has the maximum number of valid transformations, i.e., 50 out of 70 transformations are valid. *Group 20* has the minimum number of valid transformations, i.e., only 16 out of 70 transformations are valid. We observe that all transformations *Blur* are valid across 18 groups. On the contrary, five transformations, namely Rotation_24, Rotation_27, Shear_0.4, Shear_0.5, and Shear_0.6 were invalid across all groups as they failed to meet our criterion (Po-Ps<=0.1 for all three models).

2) Inconsistent Behavior Detection Results of t-way Tests

In this section, we present the results of t-way synthetic images. We generate 2-way tests based on the set of valid transformations identified from the previous step for each group. Each test represents a combination of transformations that could be used to generate synthetic image. Then, we execute the 2-way tests in three subject models to identify the number of consistent and inconsistent behaviors among three DNN models.

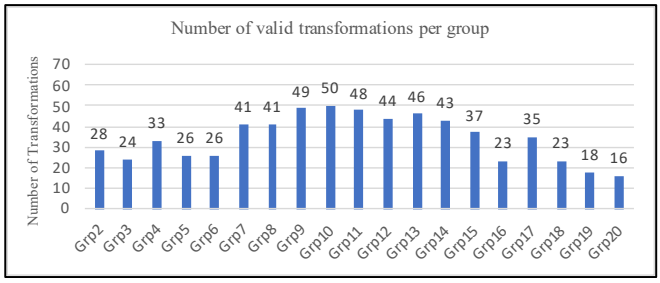

FIGURE 2 – NUMBER OF VALID TRANSFORMATIONS FOR EACH GROUP

Results for Case 1: In this case, we assume that the original seed image and t-way synthetic image share the ground-truth value. Figures 3, 4, and 5 present the t-way test results for threshold values of 0.1, 0.2, and 0.3, respectively. Recall that we generate t-way tests per group, and the total number of t-way tests varies among the groups. Therefore, we present our results as a percentage of t-way tests that exhibit consistent behavior. The x-axis represents the group number. The y-axis represents the percentage of t-way tests that exhibit consistent behavior. The last column in Table III presents the total number of t-way tests generated for each group.

For a threshold value of 0.1, our results indicate that *Rambo* is less prone to inconsistent behavior among the three subject models. As Figure 3 suggests, for eight groups (3, 7, 9, 12, 13, 15, 16, and 20), t-way tests executed with the *Rambo* model do not display any inconsistent behavior (all tests result in a passing state). In addition to this, in groups 5, 6, 14, and 18, more than 90% of the t-way tests executed with the *Rambo* model result in a consistent behavior. Apart from Group 10, the *Rambo* model exhibits a better prediction performance than the other two models. In the case of *Chauffeur*, more than 50% of t-way tests generated for seven groups (2, 7, 8, 9, 12, 14, 18) results in an inconsistent behavior; the lowest being Group 12 with a meager 16% of tests resulting in a consistent behavior. On the contrary, for the same group (Group 12), 96% of the t-way tests result in a consistent behavior for the *Rambo model*. Our results suggest that the *Autumn* model exhibits a mixed performance. In a few cases (Groups 6 and 10), more than 90% of t-way tests result in a consistent behavior state. On the contrary, for six groups (Group 2, 3, 8, 17, 18, 20), the *Autumn* model produces an inconsistent behavior for more than 50% of the t-way tests.

Figures 4 and 5 suggest an increase in the threshold value results in better performance, i.e., a higher number of consistent behaviors across three models. In the case of Rambo, with a threshold of 0.2, in most cases, all t-way tests generated exhibit a consistent behavior (16 out of 19 groups). However, for the rest of the two models, we observe that more than 25% of t-way tests still result in inconsistent behavior for some groups. For example, group 14, 17, 18, and 20 for Autumn, group 2, 8, 9, 12, 16, and 18 for Chauffeur (threshold 0.2). We observe a similar pattern for threshold 0.3. Overall, the results suggest that the *Rambo* model exhibits better performance than the other two models.

Results for Case 2: Recall that in this case, the original and t-way synthetic images might have a different ground-truth value. We evaluate the t-way test results with three threshold values: 0.1 (2.5°), 0.2(5°), and 0.3 (7.5°). Table III presents the results. The first column lists the group number. The next three columns present the number of t-way tests exhibiting inconsistent behavior for thresholds of 0.1, 0.2, and 0.3, respectively. The last column presents the total number of t-way tests for each group.

The result suggests that t-way tests can detect a significant number of inconsistent behaviors across different thresholds. For a threshold of 0.1, in 18 (out of 19) groups, 50% or more tests result in inconsistent behavior. In 12 (out of 19) groups more than 90% of tests results in inconsistent behavior.

Our results indicate that an increase in the threshold value results in a decrease in the number of inconsistent behaviors. This is as expected. With a threshold of 0.3 (7.5°), for four groups (Group 5, Group 12, Group 13, and Group 15), less than 3% of tests resulted in inconsistent behavior. This indicates that a further increase in threshold might result in a large number of false negatives. Therefore, we did not consider a threshold value that is larger than 0.3.

Overall, the results suggest that t-way test set are effective in identifying model inconsistencies. We acknowledge that both Case 1 and Case 2 have limitations. In Case 1, in some scenarios, determining the ground truth for a synthetic image generated from a t-way test set can be a challenging task (lack of test oracle). In Case 2, given the nature of differential testing, a model inconsistency can be detected only if (1) there exist at least two or more models implementing the same functionality, and (2) at least one model producing a different result. A practitioner shall choose between Case 1 and Case 2 based on their domain knowledge.

Overall, the results suggest that t-way tests are effective in identifying model inconsistencies.

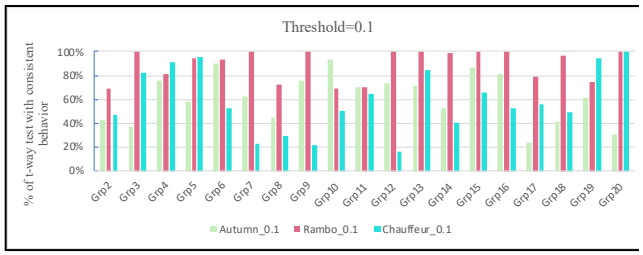

FIGURE 3 - T-WAY RESULTS FOR THRESHOLD 0.1 (CASE #1)

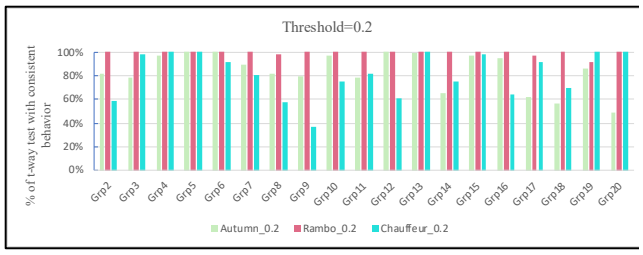

FIGURE 4- T-WAY RESULTS FOR THRESHOLD 0.2 (CASE #1)

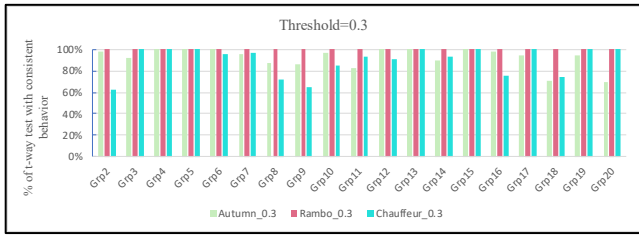

FIGURE 5- T-WAY RESULTS FOR THRESHOLD 0.3 (CASE #1)

3) T-way tests and their impact on Neuron Coverage

In this section, we present the neuron coverage achieved by t-way tests for the Rambo model. The Rambo model consists of 3 CNN sub-models referred to as S1, S2, and S3, and they consist of a total of 1625, 3801, and 13473 neurons, respectively [36]. Overall, the Rambo model consists of 18899 neurons.

Table IV presents the neuron coverage for the seed images (baseline). The results indicate that most of the seed images (17 out of 19) cover approximately 10% of the total neurons, while Group 10 and Group 11 cover 73.29% and 71.78% of the total neurons, respectively.

Next, we present the neuron coverage achieved by the t-way tests in Figures 6, 7, 8, and 9. The x-axis represents the group number. The y-axis represents the percentage of additional neurons covered by the t-way tests compared to their respective baseline. Results suggest that t-way tests result in a significant increase in neuron coverage. In the case of S1, we notice a moderate increase in the additional number of neurons covered compared to the baseline. The result presented in Table IV indicates, amongst the nineteen groups, the seed image representing Group 17 achieves the least coverage for S1 with 460 neurons. The t-way tests generated for group 17 cover an additional 25% of neurons (113 neurons) compared to its baseline. Similarly, in sub-model S2, across groups, we notice a substantial number of additional neurons covered by the t-way tests; seven groups covering more than 50% of additional neurons compared to their respective baseline.

Group Number	***# of inconsistent behaviors per threshold***			***Total # of Tests***
	0.1	***0.2***	***0.3***	
2	119	102	69	121
3	107	89	53	110
4	100	45	13	110
5	86	29	0	121
6	102	65	27	102
7	120	86	23	122
8	114	95	55	121
9	109	95	75	121
10	96	54	27	121
11	66	39	27	122
12	91	15	3	126
13	95	30	4	121
14	102	85	36	122
15	44	7	0	121
16	121	118	85	121
17	121	110	78	121
18	54	47	35	55
19	55	52	34	55
20	30	23	18	33

TABLE III – NUMBER OF INCONSISTENT BEHAVIOR IDENTIFIED BY T-WAY TESTS (CASE # 2)

We observe that t-way tests achieve a significant increase in neuron coverage for sub-model S3. Out of 19 groups, t-way tests generated for eleven groups achieve more than one hundred percent increase in cumulative neuron coverage; six groups (Group 2, 7, 8, 9 12, 13) achieve more than ten times increase in cumulative neuron coverage. On the contrary, t-way tests for two groups - Group 10 and Group 11 cover a significantly lesser number of additional neurons. This can be explained as follows: for sub-model S3, the seed images representing Group 10 and 11 cover 95.91% and 94.01% neurons. Hence, their respective t-way tests result in a marginal increase in neuron coverage.

Overall, the result suggests that t-way tests increase the neuron coverage significantly. In some cases, the results suggest a smaller number of t-way tests (120 tests) can cover more than ten times of additional set of neurons.

Group Number	*Number of Covered Neurons*			
	S1	S2	S3	Total
2	500	449	802	1751
3	501	452	1113	2066
4	497	416	722	1635
5	501	428	827	1756
6	496	445	718	1659
7	492	433	1153	2078
8	485	461	778	1724
9	483	438	795	1716
10	468	461	12923	13852
11	475	424	12667	13566
12	463	442	960	1865
13	465	422	808	1695
14	471	430	904	1805
15	467	459	806	1732
16	466	433	1224	2123
17	460	466	822	1748
18	480	456	1176	2112
19	486	422	3801	2118
20	469	447	1189	2105

TABLE IV – NEURON COVERAGE OF SEED IMAGES (RAMBO)

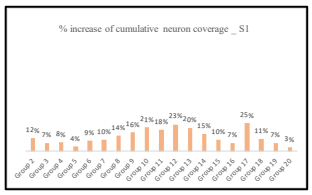

FIGURE 6 –S1

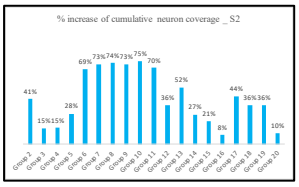

FIGURE 7 – S2

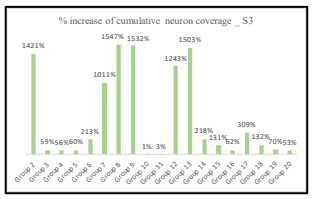

FIGURE 8 – S3

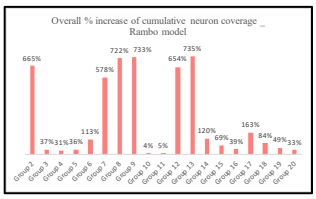

FIGURE 9 – RAMBO MODEL

We acknowledge that the neuron coverage results are unavailable for the remaining two models due to time limitations. On executing the Chauffeur model with a batch of 100 images, the current version of the framework, on average takes 18 minutes to measure the neuron coverage. So, t-way tests for a group (with 121 tests) takes around [(121*18)/60 = 36 hours]. It will take weeks to complete the coverage measurement for all nineteen groups. We plan to study the impact of t-way tests on neuron coverage of Autumn and Chauffer model as a part of future work. Also, we plan to investigate the correlation between neuron coverage and fault detection as a part of future work.

I. Threats to validity

Threats to external validity occur when the results from our experiments could not be generalized to other subjects. The DNN models used in our study have been used in other studies [36, 44, 46, 15]. All three DNN models used in our experiments have different architectures, thus alleviating the risk of lack of DNN architectures (representativeness) used in our study.

Threats to internal validity are factors that may be responsible for the experimental results, without our knowledge. To mitigate the risk of human errors, we tried to automate as many tasks as possible, from generating synthetic images to executing the tests. Also, we have manually checked some of the results whenever any inconsistent or surprising results occur. For example, out of 121 tests generated for a seed image (group 8), 72.73% percentage of tests results in a consistent behavior for Rambo model. In contrast, the other two models had less than 50% of the tests resulting in a consistent behavior. In such scenario, we manually verified the results by analyzing the log file.

V. RELATED WORK

We first discuss the existing work related to testing DNN based software systems. Traditional testing techniques such as coverage-guided testing [26, 36, 42], concolic testing [32], mutation testing [24], differential testing [27], combinatorial testing [9, 23] have been applied to test DNN models. We focus on the existing work reported in applying combinatorial testing (CT) to test machine learning systems as they are most relevant to our work.

Ma et al. proposed DeepCT, a combinatorial testing coverage guided test generation technique to test the robustness of the DNN model [23]. DeepCT follows a white box testing approach by testing the interactions of the neurons within each layer in the DNN. Similarly, Chen et al. apply variable strength combinatorial testing to test DNN models. They propose three different methods to construct variable strength-based CT tests and study their effect on interactions between pre-layer and post-layer neurons [9]. In contrast, we apply CT as a black-box approach to generate test images and detect potential predictions errors of DNN models used in autonomous vehicle software systems.

Li et al. proposed an ontology-based test generation framework for testing autonomous driving systems [22]. In Step 1, they construct an ontology based on the autonomous driving domain. In Step 2, they convert an ontology to a combinatorial test input model using conversion algorithms. Next, based on the test input model, they generate abstract tests that are used to create concrete tests. In Step 3, they execute the concrete tests and evaluate their results. Gladisch et al. proposed a combinatorial testing approach to generate a test dataset for testing perception functions [14]. They use SCODE [12] to convert a domain model to an input test model for PICT, a pair-wise test generation tool [17]. Using PICT, they generate abstract test cases that are later converted to concrete tests (test images).

Similar to [14, 22], we also develop an input parameter model (IPM), generate abstract t-way tests (based on the IPM), generate, execute and evaluate the concrete tests. Our work differs in the following way: Li et al. develop an input test model based on the road parameters such as slop, surface, and lane type. Gladisch et al. generate input test model based on the traffic scenarios such as daytime, sky conditions, rain, reflection on road etc. In contrast, we develop an IPM based on the seven image transformations techniques namely blur, brightness, contrast, rotation, scaling, shearing and translation.

Next, we discuss the existing literature on testing autonomous vehicle software systems. A significant amount of work has been reported on testing autonomous vehicle software systems [11, 13, 15, 19, 25, 27, 36, 37, 46, 47, 48]. Pei et al. proposed a technique to generate synthetic test inputs using a joint optimization problem to test DNN models used

in autonomous vehicle systems [27]. Tian et al. proposed an approach to generate test inputs (synthetic images) by simple image transformations [36]. Yan et al. presented an approach that generates tests by Adaptive Random Testing (ART) technique and uses an Adaptive Random Testing for Deep Learning Systems (ARTDL) algorithm that selects test input using a distance metric known as Feature-based Euclidean Distance (FED) to test the model under test [44]. Zhang et al. proposed an unsupervised image-to-image transformations framework based on Generative Adversarial Network (GAN) that generates synthesized test inputs that mimics two weather conditions, namely snow, and rain, to test the DNN model [46]. Haq et al. presented an empirical comparison of offline testing (testing the DNN model as an individual component) and online testing (testing the DNN model as a part of a software system) of DNN models used in autonomous driving systems [15].

Similar to our work, existing work reported in [15, 27, 36, 44, 46] have used the Udacity driving challenge-2 datasets to evaluate their respective approaches. Also, our work is similar to [36, 44, 46], in terms of generating test inputs by image transformations and testing and evaluating the DNN models using metamorphic relations. However, our work differs in the following way. Tian et al. [36] primarily study the impact of synthesized images (generated by combining different transformations) on the neuron coverage. In contrast, our work focusses on evaluating the impact of synthesized images on the model's prediction. The work presented in [15] compared the offline and online testing of DNN systems. It investigated the possibility of testing DNN's by replacing the original dataset with simulator-generated datasets. In contrast, our work explores the possibility of generating test inputs using a combinatorial testing approach to detect prediction errors in DNN models. Zhang et al. use an unsupervised network that uses GAN to generate synthesized test inputs that mimic different weather conditions [46]. Compared to [44, 46], our work is focused on generating tests using a combinatorial testing approach, i.e., generating synthesized images by combining different images transformations. To the best of our knowledge, ours is the first work that applies combinatorial testing techniques to generate t-way synthetic images for testing DNN models used in autonomous driving software systems. We also note that there is a significant number of existing studies in literature, and we refer the reader to [45] for a comprehensive report on existing work on testing machine learning systems.

VI. Conclusion and Future Work

In this paper, we present a combinatorial testing-based approach to systematically generate test images to test DNN models used in the autonomous driving systems. We begin our approach, by applying basic image transformations on the seed image (original) and identifying a set of transformations that do not change the ground truth of the image being transformed as valid transformations. Then, based on the valid transformations, we develop the IPM and generate t-way tests each of which is applied to the seed image to generate an synthetic image. We identify inconsistent behaviors of DNN models in two scenarios: (1) the original and synthetic image share the ground truth and (2) the original and synthetic image does not share the ground truth.

We performed an experimental evaluation of our approach with three publicly available pre-trained DNN models and datasets from the Udacity self-driving challenge. Our results indicate, for scenario 1, *Rambo* model exhibits a better performance, i.e., less prone to inconsistent behavior, compared to the other two models. For scenario 2, synthetic images generated by combining a set of image transformations (t-way tests) can successfully identify inconsistent behavior among models. With a threshold of 0.1, more than 90% of test cases from 12 groups result in an inconsistent behavior.

Result suggests t-way tests significantly increases the neuron coverage for the *Rambo* model. Out of the 19 groups, synthetic images generated for 17 groups, result in a moderate to significant increase in cumulative neuron coverage; nine groups (Group 2, 6, 7, 8, 9, 12, 13, 14, 17) achieves more than one hundred percent increase in cumulative neuron coverage. Given the time-intensive nature of the measurement process, we are unable to measure the neuron coverage for the remaining two models. We plan to complete the measurement as a part of future work.

This is part of our larger effort in applying combinatorial testing to test DNN based systems. We plan to include additional weather-based transformations such as rain, fog, smog, and shadows to generate test images. We hope to leverage the insights gained from this study to refine our input parameter model, develop realistic and meaningful constraints and thus generating more effective t-way tests to test DNN based systems. Also, we plan to extend this work by investigating how the combinatorial testing-based approach can be adopted in testing different versions of the DNN models in regression testing.

Acknowledgment

This work is supported by research grant (70NANB18H207) from Information Technology Lab of National Institute of Standards and Technology (NIST).

Disclaimer: Certain software products are identified in this document. Such identification does not imply recommendation by the NIST, nor does it imply that the products identified are necessarily the best available for the purpose.

References

[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Kudlur, M. (2016). Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)* (pp. 265-283).

[2] Advanced Combinatorial Testing System (ACTS) | CSRC, https://csrc.nist.gov/projects/automated-combinatorial-testing-for-software/downloadable-tools, Accessed: 2021-02-28

[3] ARiSe-Lab/deepTest: A systematic testing tool: https://github.com/ARiSE-Lab/deepTest, Accessed: 2020-10-12

[4] "Autumn-model:" https://github.com/udacity/self-driving-car/tree/master/steering-models/community-models/autumn, Accessed: 2020-09-12

[5] Barr, E. T., Harman, M., McMinn, P., Shahbaz, M., & Yoo, S. (2014). The oracle problem in software testing: A survey. *IEEE transactions on software engineering*, *41*(5), 507-525.

[6] Chandrasekaran, J., Feng, H., Lei, Y., Kuhn, D. R., & Kacker, R. (2017, March). Applying combinatorial testing to data mining algorithms. In *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)* (pp. 253-261). IEEE.

[7] "Chauffeur-model:" https://github.com/udacity/self-driving-car/tree/master/steering-models/community-models/chauffeur, Accessed: 2020-09-12

[8] Chen, T. Y., Cheung, S. C., & Yiu, S. M. (2020). Metamorphic testing: a new approach for generating next test cases. *arXiv preprint arXiv:2002.12543*.

[9] Chen, Y., Wang, Z., Wang, D., Fang, C., & Chen, Z. (2019, April). Variable strength combinatorial testing for deep neural networks. In *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)* (pp. 281-284). IEEE.

[10] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. The AETG system: An approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*, 23(7):437–444, July 1997

[11] Dreossi, T., Ghosh, S., Sangiovanni-Vincentelli, A., & Seshia, S. A. (2017). Systematic testing of convolutional neural networks for autonomous driving. *arXiv preprint arXiv:1708.03309*.

[12] ETAS GmbH. SCODE-ANALYZER Software for describing and visualizing complex closed-loop control systems, 2019. https://www.etas.com/scode.

[13] Gambi, A., Mueller, M., & Fraser, G. (2019, July). Automatically testing self-driving cars with search-based procedural content generation. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis* (pp. 318-328).

[14] Gladisch, C., Heinzemann, C., Herrmann, M., & Woehrle, M. (2020). Leveraging combinatorial testing for safety-critical computer vision datasets. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops* (pp. 324-325).

[15] Haq, F. U., Shin, D., Nejati, S., & Briand, L. C. (2020, October). Comparing Offline and Online Testing of Deep Neural Networks: An Autonomous Car Case Study. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)* (pp. 85-95). IEEE.

[16] Herbold, S., & Haar, T. (2020). Smoke Testing for Machine Learning: Simple Tests to Discover Severe Defects. *arXiv preprint arXiv:2009.01521*.

[17] Jacek Czerwonka. Pairwise testing in real world. In *24th Pacific Northwest Software Quality Conf.*, volume 200, 2006.

[18] Keras-team/Keras: Deep Learning for humans: https://github.com/keras-team/keras, Accessed: 2020-10-16

[19] Kim, J., Feldt, R., & Yoo, S. (2019, May). Guiding deep learning system testing using surprise adequacy. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)* (pp. 1039-1049). IEEE.

[20] Komanda model weight file: https://s3.amazonaws.com/udacity-sdc/steering-models/komanda/udacity-challenge2-model/FINE_TUNE_2-checkpoint-sdc-ch2-epoch5, Accessed: 2020-10-19

[21] Kuhn, D. R., Kacker, R. N., & Lei, Y. (2010). Practical combinatorial testing. *NIST special Publication*, *800*(142), 142.

[22] Li, Y., Tao, J., & Wotawa, F. (2020). Ontology-based test generation for automated and autonomous driving functions. *Information and software technology*, *117*, 106200.

[23] Ma, L., Juefei-Xu, F., Xue, M., Li, B., Li, L., Liu, Y., & Zhao, J. (2019, February). Deepct: Tomographic combinatorial testing for deep learning systems. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)* (pp. 614-618). IEEE.

[24] Ma, L., Zhang, F., Sun, J., Xue, M., Li, B., Juefei-Xu, F., ... & Wang, Y. (2018, October). Deepmutation: Mutation testing of deep learning systems. In *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)* (pp. 100-111). IEEE.

[25] Majumdar, R., Mathur, A., Pirron, M., Stegner, L., & Zufferey, D. (2019). Paracosm: A language and tool for testing autonomous driving systems. *arXiv preprint arXiv:1902.01084*.

[26] Odena, A., Olsson, C., Andersen, D., & Goodfellow, I. (2019, May). Tensorfuzz: Debugging neural networks with coverage-guided fuzzing. In *International Conference on Machine Learning* (pp. 4901-4911).

[27] Pei, K., Cao, Y., Yang, J., & Jana, S. (2017, October). Deepxplore: Automated whitebox testing of deep learning systems. In *proceedings of the 26th Symposium on Operating Systems Principles* (pp. 1-18).

[28] R. Bryce, C. J. Colbourn, M.B. Cohen, "A framework of greedy methods for constructing interaction tests," Proceedngs of the 27th International Conference on Software Engineering (ICSE), pp. 146-155, 2005

[29] "Rambo-model:" https://github.com/udacity/self-driving-car/tree/master/steering-models/community-models/rambo, Accessed: 2020-09-12

[30] Rwightman-model script file: https://github.com/udacity/self-driving-car/blob/master/steering-models/evaluation/rwightman.py, Accessed: 2021-01-20.

[31] Self-driving-car-Results-Dropbox, https://tinyurl.com/y2s6qxeo, Accessed: 2021-01-20.

[32] Sun, Y., Wu, M., Ruan, W., Huang, X., Kwiatkowska, M., & Kroening, D. (2018, September). Concolic testing for deep neural networks. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*(pp. 109-119).

[33] Team, T. T. D., Al-Rfou, R., Alain, G., Almahairi, A., Angermueller, C., Bahdanau, D., ... & Belopolsky, A. (2016). Theano: A Python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*.

[34] Tesla driver dies in first fatal crash while using autopilot mode: https://www.theguardian.com/technology/2016/jun/30/tesla-autopilot-death-self-driving-car-elon-musk, Accessed: 2020-10-19.

[35] Testing-AI-Systems, https://github.com/cjaganmohan/Testing-AI-Systems, Accessed: 2021-01-20.

[36] Tian, Y., Pei, K., Jana, S., & Ray, B. (2018, May). Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering* (pp. 303-314).

[37] Tuncali, C. E., Fainekos, G., Ito, H., & Kapinski, J. (2018, June). Simulation-based adversarial test generation for autonomous vehicles with machine learning components. In *2018 IEEE Intelligent Vehicles Symposium (IV)* (pp. 1555-1562). IEEE.

[38] Uber's self-driving car didn't know pedestrians could jaywalk: https://www.wired.com/story/ubers-self-driving-car-didnt-know-pedestrians-could-jaywalk/, Accessed: 2020-09-27

[39] "Udacity Challenge 2 – Test Dataset:" https://github.com/udacity/self-driving-car/blob/master/challenges/challenge-2/CH2_final_evaluation.csv, Accessed: 2020-09-12

[40] "Udacity self-driving challenge 2," https://github.com/udacity/self-driving-car/tree/master/steering-models/community-models, Accessed: 2020-09-12

[41] Wicker, M., Huang, X., & Kwiatkowska, M. (2018, April). Feature-guided black-box safety testing of deep neural networks. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (pp. 408-426). Springer, Cham.

[42] Xie, X., Ma, L., Juefei-Xu, F., Xue, M., Chen, H., Liu, Y., ... & See, S. (2019, July). Deephunter: A coverage-guided fuzz testing framework for deep neural networks. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis* (pp. 146-157).

[43] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence. Ipog/ipog-d: efficient test generation for multi-way combinatorial testing. *Software Testing, Verification and Reliability*, 18(3):125–148, 2008.

[44] Yan, M., Wang, L., & Fei, A. (2019). ARTDL: Adaptive Random Testing for Deep Learning Systems. *IEEE Access*, *8*, 3055-3064.

[45] Zhang, J. M., Harman, M., Ma, L., & Liu, Y. (2020). Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering*.

[46] Zhang, M., Zhang, Y., Zhang, L., Liu, C., & Khurshid, S. (2018, September). DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems. In *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 132-142). IEEE.

[47] Zhou, H., Li, W., Zhu, Y., Zhang, Y., Yu, B., Zhang, L., & Liu, C. (2018). Deepbillboard: Systematic physical-world testing of autonomous driving systems. *arXiv preprint arXiv:1812.10812*.

[48] Zhou, Z. Q., & Sun, L. (2019). Metamorphic testing of driverless cars. *Communications of the ACM*, *62*(3), 61-67.

[49] 2015. Open Source Computer Vision Library. https://github.com/itseez/opencv (2015).