We want to find images that trigger differences in the outputs that suggest BUGS, ☺

*

DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars

Yuchi Tian
University of Virginia
yuchi@virginia.edu

Kexin Pei
Columbia University
kpei@cs.columbia.edu

Suman Jana
Columbia University
suman@cs.columbia.edu

Baishakhi Ray
University of Virginia
rayb@virginia.edu

ABSTRACT

Recent advances in Deep Neural Networks (DNNs) have led to the development of DNN-driven autonomous cars that, using sensors like camera, LiDAR, etc., can drive without any human intervention. Most major manufacturers including Tesla, GM, Ford, BMW, and Waymo/Google are working on building and testing different types of autonomous vehicles. The lawmakers of several US states including California, Texas, and New York have passed new legislation to fast-track the process of testing and deployment of autonomous vehicles on their roads.

However, despite their spectacular progress, DNNs, just like traditional software, often demonstrate incorrect or unexpected corner-case behaviors that can lead to potentially fatal collisions. Several such real-world accidents involving autonomous cars have already happened including one which resulted in a fatality. Most existing testing techniques for DNN-driven vehicles are heavily dependent on the manual collection of test data under different driving conditions which become prohibitively expensive as the number of test conditions increases.

In this paper, we design, implement, and evaluate DeepTest, a systematic testing tool for automatically detecting erroneous behaviors of DNN-driven vehicles that can potentially lead to fatal crashes. First, our tool is designed to automatically generated test cases leveraging real-world changes in driving conditions like rain, fog, lighting conditions, etc. DeepTest systematically explore different parts of the DNN logic by generating test inputs that maximize the numbers of activated neurons. DeepTest found thousands of erroneous behaviors under different realistic driving conditions (e.g., blurring, rain, fog, etc.) many of which lead to potentially fatal crashes in three top performing DNNs in the Udacity self-driving car challenge.

CCS CONCEPTS

• **Software and its engineering** → **Software testing and debugging**; • **Security and privacy** → *Software and application security*; • **Computing methodologies** → *Neural networks*;

KEYWORDS

deep learning, testing, self-driving cars, deep neural networks, autonomous vehicle, neuron coverage

ACM Reference Format:
Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars. In *ICSE '18: ICSE '18: 40th International Conference on Software Engineering , May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3180155.3180220

1 INTRODUCTION

Significant progress in Machine Learning (ML) techniques like Deep Neural Networks (DNNs) over the last decade has enabled the development of safety-critical ML systems like autonomous cars. Several major car manufacturers including Tesla, GM, Ford, BMW, and Waymo/Google are building and actively testing these cars. Recent results show that autonomous cars have become very efficient in practice and already driven millions of miles without any human intervention [21, 36]. Twenty US states including California, Texas, and New York have recently passed legislation to enable testing and deployment of autonomous vehicles [18].

However, despite the tremendous progress, just like traditional software, DNN-based software, including the ones used for autonomous driving, often demonstrate incorrect/unexpected corner-case behaviors that can lead to dangerous consequences like a fatal collision. Several such real-world cases have already been reported (see Table 1). As Table 1 clearly shows, such crashes often happen under rare previously unseen corner cases. For example, the fatal Tesla crash resulted from a failure to detect a white truck against the bright sky. The existing mechanisms for detecting such erroneous behaviors depend heavily on manual collection of labeled test data or ad hoc, unguided simulation [11, 20] and therefore miss numerous corner cases. Since these cars adapt behavior based on their environment as measured by different sensors (e.g., camera, Infrared obstacle detector, etc.), the space of possible inputs is extremely large. Thus, unguided simulations are highly unlikely to find many erroneous behaviors.

At a conceptual level, these erroneous corner-case behaviors in DNN-based software are analogous to logic bugs in traditional software. Similar to the bug detection and patching cycle in traditional software development, the erroneous behaviors of DNNs, once detected, can be fixed by adding the error-inducing inputs to the training data set and also by possibly changing the model structure/parameters. However, this is a challenging problem, as noted by large software companies like Google and Tesla that have already deployed machine learning techniques in several production-scale

ICSE '18, May 27-June 3, 2018, Gothenburg, Sweden
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5638-1/18/05...$15.00
https://doi.org/10.1145/3180155.3180220

Table 1: Examples of real-world accidents involving autonomous cars

	Reported Date	Cause	Outcome	Comments
Hyundai Competition [4]	December, 2014	Rain fall	Crashed while testing	"The sensors failed to pick up street signs, lane markings, and even pedestrians due to the angle of the car shifting in rain and the direction of the sun" [4]
Tesla autopilot mode [17]	July, 2016	Image contrast	Killed the driver	"The camera failed to recognize the white truck against a bright sky" [23]
Google self-driving car [12]	February, 2016	Failed to estimate speed	Hit a bus while shifting lane	"The car assumed that the bus would yield when it attempted to merge back into traffic" [12]

systems including self-driving car, speech recognition, image search, etc. [22, 73].

Our experience with traditional software has shown that it is hard to build robust safety-critical systems only using manual test cases. Moreover, the internals of traditional software and new DNN-based software are fundamentally different. For example, unlike traditional software where the program logic is manually written by the software developers, DNN-based software automatically learns its logic from a large amount of data with minimal human guidance. In addition, the logic of a traditional program is expressed in terms of control flow statements while DNNs use weights for edges between different neurons and nonlinear activation functions for similar purposes. These differences make automated testing of DNN-based software challenging by presenting several interesting and novel research problems.

First, traditional software testing techniques for systematically exploring different parts of the program logic by maximizing branch/code coverage is not very useful for DNN-based software as the logic is not encoded using control flow [70]. Next, DNNs are fundamentally different from the models (e.g., finite state machines) used for modeling and testing traditional programs. Unlike the traditional models, finding inputs that will result in high model coverage in a DNN is significantly more challenging due to the non-linearity of the functions modeled by DNNs. Moreover, the Satisfiability Modulo Theory (SMT) solvers that have been quite successful at generating high-coverage test inputs for traditional software are known to have trouble with formulas involving floating-point arithmetic and highly nonlinear constraints, which are commonly used in DNNs. In fact, several research projects have already attempted to build custom tools for formally verifying safety properties of DNNs. Unfortunately, none of them scale well to real-world-sized DNNs [48, 51, 71]. Finally, manually creating specifications for complex DNN systems like autonomous cars is infeasible as the logic is too complex to manually encode as it involves mimicking the logic of a human driver.

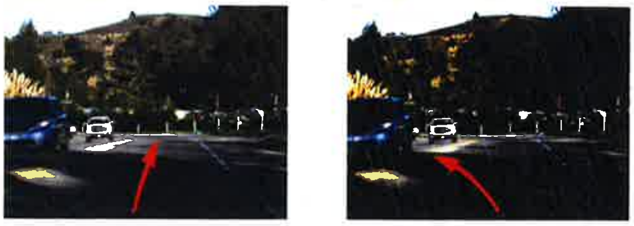
In this paper, we address these issues and design a systematic testing methodology for automatically detecting erroneous behaviors of DNN-based software of self-driving cars. First, we leverage the notion of neuron coverage (i.e., the number of neurons activated by a set of test inputs) to systematically explore different parts of the DNN logic. We empirically demonstrate that changes in neuron coverage are statistically correlated with changes in the actions of self-driving cars (e.g., steering angle). Therefore, neuron coverage can be used as a guidance mechanism for systemically exploring different types of car behaviors and identify erroneous behaviors. Next, we demonstrate that different image transformations that mimic real-world differences in driving conditions like changing contrast/brightness, rotation of the camera result in activation of different sets of neurons in the self-driving car DNNs. We show that by combining these image transformations, the neuron coverage can be increased by 100% on average compared to the coverage achieved by manual test inputs. Finally, we use transformation-specific metamorphic relations between multiple executions of the tested DNN (e.g., a car should behave similarly under different lighting conditions) to automatically detect erroneous corner case behaviors. We found thousands of erroneous behaviors across the three top performing DNNs in the Udacity self-driving car challenge [15].

?

The key contributions of this paper are:

- We present a systematic technique to automatically synthesize test cases that maximizes neuron coverage in safety-critical DNN-based systems like autonomous cars. We empirically demonstrate that changes in neuron coverage correlate with changes in an autonomous car's behavior.
- We demonstrate that different realistic image transformations like changes in contrast, presence of fog, etc. can be used to generate synthetic tests that increase neuron coverage. We leverage transformation-specific metamorphic relations to automatically detect erroneous behaviors. Our experiments also show that the synthetic images can be used for retraining and making DNNs more robust to different corner cases.
- We implement the proposed techniques in DeepTest, to the best of our knowledge, the first systematic and automated testing tool for DNN-driven autonomous vehicles. We use DeepTest to systematically test three top performing DNN models from the Udacity driving challenge. DeepTest found thousands of erroneous behaviors in these systems many of which can lead to potentially fatal collisions as shown in Figure 1.
- We have made the erroneous behaviors detected by DeepTest available at https://deeplearningtest.github.io/deepTest/. We also plan to release the generated test images and the source of DeepTest for public use.

Camera IN → Steering angle OUT



1.1 original

1.2 with added rain

?

Figure 1: A sample dangerous erroneous behavior found by DeepTest in the *Chauffeur* DNN.

What is dangerous here?

2 BACKGROUND

2.1 Deep Learning for Autonomous Driving

The key component of an autonomous vehicle is the perception module controlled by the underlying Deep Neural Network (DNN) [14, 19]. The DNN takes input from different sensors like camera, light detection and ranging sensor (LiDAR), and IR (infrared) sensor that measure the environment and outputs the steering angle, braking, etc. necessary to maneuver the car safely under

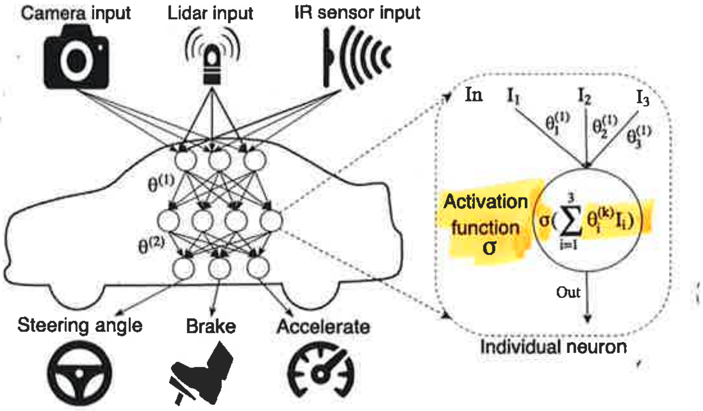


Figure 2: A simple autonomous car DNN that takes inputs from camera, light detection and ranging sensor (LiDAR), and IR (infrared) sensor, and outputs steering angle, braking decision, and acceleration decision. The DNN shown here essentially models the function $\sigma(\theta^{(2)} \cdot \sigma(\theta^{(1)} \cdot \boldsymbol{x}))$ where $\boldsymbol{\theta}$s represent the weights of the edges and σ is the activation function. The details of the computations performed inside a single neuron are shown on the right.

current conditions as shown in Figure 2. In this paper, we focus on the camera input and the steering angle output.

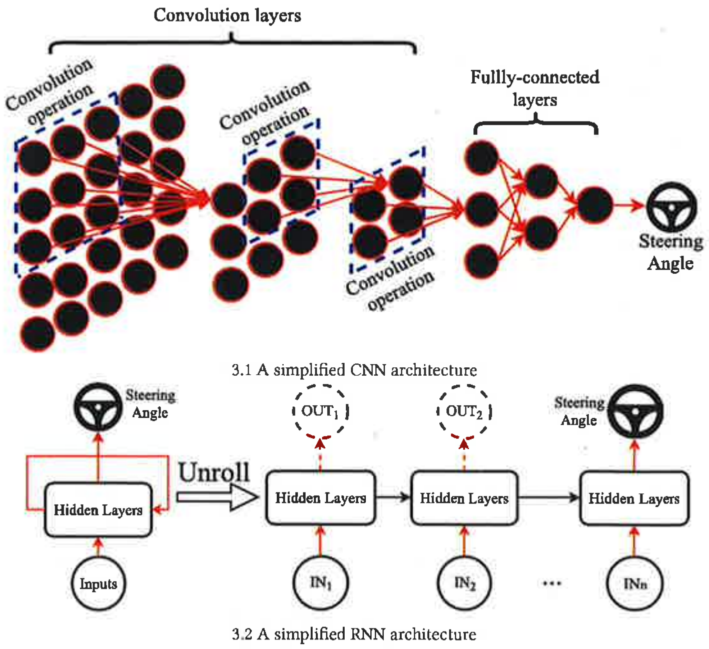
A typical feed-forward DNN is composed of multiple processing *layers* stacked together to extract different representations of the input [30]. Each layer of the DNN increasingly abstracts the input, e.g., from raw pixels to semantic concepts. For example, the first few layers of an autonomous car DNN extract low-level features such as edges and directions, while the deeper layers identify objects like stop signs and other cars, and the final layer outputs the steering decision (e.g., turning left or right).

Each layer of a DNN consists of a sequence of individual computing units called *neurons*. The neurons in different layers are connected with each other through edges. Each edge has a corresponding weight (θs in Figure 2). Each neuron applies a nonlinear *activation function* on its inputs and sends the output to the subsequent neurons as shown in Figure 2. Popular activation functions include ReLU (Rectified Linear Unit) [61], sigmoid [58], etc. The edge weights of a DNN is inferred during the training process of the DNN based on labeled training data. Most existing DNNs are trained with gradient descent using backpropagation [72]. Once trained, a DNN can be used for prediction without any further changes to the weights. For example, an autonomous car DNN can predict the steering angle based on input images.

Figure 2 illustrates a basic DNN in the perception module of a self-driving car. Essentially, the DNN is a sequence of linear transformations (*e.g.*, dot product between the weight parameters $\boldsymbol{\theta}$ of each edge and the output value of the source neuron of that edge) and nonlinear activations (e.g., ReLU in each neuron). Recent results have demonstrated that a well-trained DNN f can predict the steering angle with an accuracy close to that of a human driver [31].

2.2 Different DNN Architectures

Most DNNs used in autonomous vehicles can be categorized into two types: (1) Feed-forward Convolutional Neural Network (CNN), and (2) Recurrent neural network (RNN). The DNNs we tested (see Section 4) include two CNNs and one RNN. We provide a brief description of each architecture below and refer the interested readers to [39] for more detailed descriptions.



3.1 A simplified CNN architecture

3.2 A simplified RNN architecture

Figure 3: (Upper row) A simplified CNN architecture with a convolution kernel shown on the top-left part of the input image. The same filter (edges with same weights) is then moved across the entire input space, and the dot products are computed between the edge weights and the outputs of the connected neurons. (Lower row) A simplified RNN architecture with loops in its hidden layers. The unrolled version on the right shows how the loop allows a sequence of inputs (*i.e.* images) to be fed to the RNN and the steering angle is predicted based on all those images.

CNN architecture. The most significant difference between a CNN and a fully connected DNN is the presence of a *convolution layer*. The neurons in a convolution layer are connected only to some of the neurons in the next layer and multiple connections among different neurons share the same weight. The sets of connections sharing the same weights are essentially a convolution kernel [50] that applies the same convolution operation on the outputs of a set of neurons in the previous layer. Figure 3 (upper row) illustrates the convolution operations for three convolution layers. This simplified architecture is similar to the ones used in practice [31].

Convolution layers have two major benefits. First, they greatly reduce the number of trainable weights by allowing sharing of weights among multiple connections and thus significantly cut down the training time. Second, the application of convolution kernels is a natural fit for image recognition as it resembles the human visual system which extracts a layer-wise representation of visual input [50, 53].

RNN architecture. RNNs, unlike CNNs, allow *loops* in the network [49]. Specifically, the output of each layer is not only fed to the following layer but also flow back to the previous layer. Such arrangement allows the prediction output for previous inputs (e.g., previous frames in a video sequence) to be also considered in predicting current input. Figure 3 (lower row) illustrates a simplified version of the RNN architecture.

Similar to other types of DNNs, RNNs also leverage gradient descent with back propagation for training. However, it is well known that the gradient, when propagated through multiple loops in an RNNs, may vanish to zero or explode to an extremely large value [46] and therefore may lead to an inaccurate model. *Long* *short-term memory* (LSTM) [47], a popular subgroup of RNNs, is designed to solve this vanishing/exploding gradient problem. We encourage interested readers to refer to [47] for more details.

3 METHODOLOGY

Thats?

To develop an automated testing methodology for DNN-driven autonomous cars we must answer the following questions. (i) How do we systematically explore the input-output spaces of an autonomous car DNN? (ii) How can we synthesize realistic inputs to automate such exploration? (iii) How can we optimize the exploration process? (iv) How do we automatically create a test oracle that can detect erroneous behaviors without detailed manual specifications? We briefly describe how DeepTest addresses each of these questions below.

Same for train signal example ?

3.1 Systematic Testing with Neuron Coverage

The input-output space (i.e., all possible combinations of inputs and outputs) of a complex system like an autonomous vehicle is too large for exhaustive exploration. Therefore, we must devise a systematic way of partitioning the space into different equivalence classes and try to cover all equivalence classes by picking one sample from each of them. In this paper, we leverage neuron coverage [70] as a mechanism for partitioning the input space based on the assumption that all inputs that have similar neuron coverage are part of the same equivalence class (i.e., the target DNN behaves similarly for these inputs).

Is this true?

Neuron coverage was originally proposed by Pei *et al.* for guided differential testing of multiple similar DNNs [70]. It is defined as the ratio of unique neurons that get activated for given input(s) and the total number of neurons in a DNN:

$$Neuron\ Coverage = \frac{|Activated\ Neurons|}{|Total\ Neurons|} \quad (1)$$

An individual neuron is considered activated if the neuron's output (scaled by the overall layer's outputs) is larger than a DNN-wide threshold. In this paper, we use 0.2 as the neuron activation threshold for all our experiments.

Similar to the code-coverage-guided testing tools for traditional software, DeepTest tries to generate inputs that maximize neuron coverage of the test DNN. As each neuron's output affects the final output of a DNN, maximizing neuron coverage also increases output diversity. We empirically demonstrate this effect in Section 5.

Pei *et al.* defined neuron coverage only for CNNs [70]. We further generalize the definition to include RNNs. Neurons, depending on the type of the corresponding layer, may produce different types of output values (*i.e.* single value and multiple values organized in a multidimensional array). We describe how we handle such cases in detail below.

For all neurons in fully-connected layers, we can directly compare their outputs against the neuron activation threshold as these neurons output a single scalar value. By contrast, neurons in convolutional layers output multidimensional feature maps as each neuron outputs the result of applying a convolutional kernel across the input space [45]. For example, the first layer in Figure 3.1 illustrates the application of one convolutional kernel (of size 3×3) to the entire image (5×5) that produces a feature map of size 3×3 in the succeeding layer. In such cases, we compute the average of the output feature map to convert the multidimensional output of a neuron into a scalar and compare it to the neuron activation threshold.

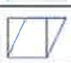
For RNN/LSTM with loops, the intermediate neurons are unrolled to produce a sequence of outputs (Figure 3.2). We treat each neuron in the unrolled layers as a separate individual neuron for the purpose of neuron coverage computation.

3.2 Increasing Coverage with Synthetic Images

Generating arbitrary inputs that maximize neuron coverage may not be very useful if the inputs are not likely to appear in the real-world even if these inputs potentially demonstrate buggy behaviors. Therefore, DeepTest focuses on generating realistic synthetic images by applying image transformations on seed images and mimic different real-world phenomena like camera lens distortions, object movements, different weather conditions, etc. To this end, we investigate nine different realistic image transformations (changing brightness, changing contrast, translation, scaling, horizontal shearing, rotation, blurring, fog effect, and rain effect). These transformations can be classified into three groups: linear, affine, and convolutional. Our experimental results, as described in Section 5, demonstrate that all of these transformations increase neuron coverage significantly for all of the tested DNNs. Below, we describe the details of the transformations.

Adjusting brightness and contrast are both linear transformations. The brightness of an image depends on how large the pixel values are for that image. An image's brightness can be adjusted by adding/subtracting a constant parameter β to each pixel's current value. Contrast represents the difference in brightness between different pixels in an image. One can adjust an image's contrast by multiplying each pixel's value by a constant parameter α.

Table 2: Different affine transformation matrices

Affine Transform	Example	Transformation Matrix	Parameters
Translation		$\begin{bmatrix}1 & 0 & t_x\\0 & 1 & t_y\end{bmatrix}$	t_x: displacement along x axis t_y: displacement along y axis
Scale		$\begin{bmatrix}s_x & 0 & 0\\0 & s_y & 0\end{bmatrix}$	s_x: scale factor along x axis s_y: scale factor along y axis
Shear		$\begin{bmatrix}1 & s_x & 0\\s_y & 1 & 0\end{bmatrix}$	s_x: shear factor along x axis s_y: shear factor along y axis
Rotation		$\begin{bmatrix}\cos q & -\sin q & 0\\\sin q & \cos q & 0\end{bmatrix}$	q: the angle of rotation

Translation, scaling, horizontal shearing, and rotation are all different types of affine transformations. An affine transformation is a linear mapping between two images that preserves points, straight lines, and planes [5]. Affine transforms are often used in image processing to fix distortions resulting from camera angle variations. In this paper, we leverage affine transformations for the inverse case, i.e., to simulate different real-world camera perspectives or movements of objects and check how robust the self-driving DNNs are to those changes.

An affine transformation is usually represented by a 2×3 transformation matrix M [6]. One can apply an affine transformation to a 2D image matrix I by simply computing the dot product of I and M, the corresponding transformation matrix. We list the transformation matrices for the four types of affine transformations (translation, scale, shear, and rotation) used in this paper in Table 2.

Blurring and adding fog/rain effects are all convolutional transformations, i.e., they perform the convolution operation on the input pixels with different transform-specific kernels. A convolution operation adds (weighted by the kernel) each pixel of the input *short-term memory* (LSTM) [47], a popular subgroup of RNNs, is designed to solve this vanishing/exploding gradient problem. We encourage interested readers to refer to [47] for more details.

3 METHODOLOGY

Thaks?

To develop an automated testing methodology for DNN-driven autonomous cars we must answer the following questions. (i) How do we systematically explore the input-output spaces of an autonomous car DNN? (ii) How can we synthesize realistic inputs to automate such exploration? (iii) How can we optimize the exploration process? (iv) How do we automatically create a test oracle that can detect erroneous behaviors without detailed manual specifications? We briefly describe how DeepTest addresses each of these questions below.

Same for train signal example ?

3.1 Systematic Testing with Neuron Coverage

The input-output space (i.e., all possible combinations of inputs and outputs) of a complex system like an autonomous vehicle is too large for exhaustive exploration. Therefore, we must devise a systematic way of partitioning the space into different equivalence classes and try to cover all equivalence classes by picking one sample from each of them. In this paper, we leverage neuron coverage [70] as a mechanism for partitioning the input space based on the assumption that all inputs that have similar neuron coverage are part of the same equivalence class (i.e., the target DNN behaves similarly for these inputs).

is this true?

Neuron coverage was originally proposed by Pei *et al.* for guided differential testing of multiple similar DNNs [70]. It is defined as the ratio of unique neurons that get activated for given input(s) and the total number of neurons in a DNN:

$$Neuron\ Coverage = \frac{|Activated\ Neurons|}{|Total\ Neurons|} \tag{1}$$

An individual neuron is considered activated if the neuron's output (scaled by the overall layer's outputs) is larger than a DNN-wide threshold. In this paper, we use 0.2 as the neuron activation threshold for all our experiments.

Similar to the code-coverage-guided testing tools for traditional software, DeepTest tries to generate inputs that maximize neuron coverage of the test DNN. As each neuron's output affects the final output of a DNN, maximizing neuron coverage also increases output diversity. We empirically demonstrate this effect in Section 5.

Pei *et al.* defined neuron coverage only for CNNs [70]. We further
generalize the d
the type of the c
output values (*i.*
multidimensiona
detail below.

For all neuror
pare their output
neurons output a
volutional layers
neuron outputs tl
the input space
lustrates the app
to the entire ima
in the succeeding
the output featu
of a neuron into
threshold.

For RNN/LSTM with loops, the intermediate neurons are unrolled to produce a sequence of outputs (Figure 3.2). We treat each neuron in the unrolled layers as a separate individual neuron for the purpose of neuron coverage computation.

3.2 Increasing Coverage with Synthetic Images

Generating arbitrary inputs that maximize neuron coverage may not be very useful if the inputs are not likely to appear in the real-world even if these inputs potentially demonstrate buggy behaviors. Therefore, DeepTest focuses on generating realistic synthetic images by applying image transformations on seed images and mimic different real-world phenomena like camera lens distortions, object movements, different weather conditions, etc. To this end, we investigate nine different realistic image transformations (changing brightness, changing contrast, translation, scaling, horizontal shearing, rotation, blurring, fog effect, and rain effect). These transformations can be classified into three groups: linear, affine, and convolutional. Our experimental results, as described in Section 5, demonstrate that all of these transformations increase neuron coverage significantly for all of the tested DNNs. Below, we describe the details of the transformations.

Adjusting brightness and contrast are both linear transformations. The brightness of an image depends on how large the pixel values are for that image. An image's brightness can be adjusted by adding/subtracting a constant parameter β to each pixel's current value. Contrast represents the difference in brightness between different pixels in an image. One can adjust an image's contrast by multiplying each pixel's value by a constant parameter α.

Table 2: Different affine transformation matrices

Affine Transform	Example	Transformation Matrix	Parameters
Translation		$\begin{bmatrix}1 & 0 & t_x\\0 & 1 & t_y\end{bmatrix}$	t_x: displacement along x axis t_y: displacement along y axis
Scale		$\begin{bmatrix}s_x & 0 & 0\\0 & s_y & 0\end{bmatrix}$	s_x: scale factor along x axis s_y: scale factor along y axis
Shear		$\begin{bmatrix}1 & s_x & 0\\s_y & 1 & 0\end{bmatrix}$	s_x: shear factor along x axis s_y: shear factor along y axis
Rotation		$\begin{bmatrix}\cos q & -\sin q & 0\\\sin q & \cos q & 0\end{bmatrix}$	q: the angle of rotation

Translation, scaling, horizontal shearing, and rotation are all different types of affine transformations. An affine transformation is
ges that preserves points, straight
sforms are often used in image
ing from camera angle variations.
transformations for the inverse
al-world camera perspectives or
ow robust the self-driving DNNs

ally represented by a 2 × 3 trans-
apply an affine transformation
computing the dot product of I
mation matrix. We list the trans-
types of affine transformations
on) used in this paper in Table 2.
fects are all convolutional trans-
convolution operation on the
rm-specific kernels. A convolu-
ne kernel) each pixel of the input

Need more than ONE coverage model, e.g. HW Design Verif.:

- code
- structural
- functional

~> All see the DUV from different angles ☺

image to its local neighbors. We use four different types of blurring filters: averaging, Gaussian, median, and bilateral [3]. We compose multiple filters provided by Adobe Photoshop on the input images to simulate realistic fog and rain effects [1, 2].

3.3 Combining Transformations to Increase Coverage

As the individual image transformations increase neuron coverage, one obvious question is whether they can be combined to further increase the neuron coverage. Our results demonstrate that different image transformations tend to activate different neurons, i.e., they can be stacked together to further increase neuron coverage. However, the state space of all possible combinations of different transformations is too large to explore exhaustively. We provide a neuron-coverage-guided greedy search technique for efficiently finding combinations of image transformations that result in higher coverage (see Algorithm 1).

Algorithm 1: Greedy search for combining image tranformations to increase neuron coverage

```
Input    :Transformations T, Seed images I
Output   :Synthetically generated test images
Variable :S: stack for storing newly generated images
          Tqueue: transformation queue

Push all seed imgs ∈ I to Stack S
genTests = $\phi$
while S is not empty do
    img = S.pop()
    Tqueue = $\phi$
    numFailedTries = 0
    while numFailedTries ≤ maxFailedTries do
        if Tqueue is not empty then
            T1 = Tqueue.dequeue()
        else
            Randomly pick transformation T1 from T
        end
        Randomly pick parameter P1 for T1
        Randomly pick transformation T2 from T
        Randomly pick parameter P2 for T2
        newImage = ApplyTransforms(image, T1, P1, T2, P2)
        if covInc(newimage) then
            Tqueue.enqueue(T1)
            Tqueue.enqueue(T2)
            UpdateCoverage()
            genTest = genTests ∪ newimage S.push(newImage)
        else
            numFailedTries = numFailedTries + 1
        end
    end
end
return genTests
```

trial... & error approach (bruteforce?)

The algorithm takes a set of seed images I, a list of transformations T and their corresponding parameters as input. The key idea behind the algorithm is to keep track of the transformations that successfully increase neuron coverage for a given image and prioritize them while generating more synthetic images from the given image. This process is repeated in a depth-first manner to all images.

3.4 Creating a Test Oracle with Metamorphic Relations

One of the major challenges in testing a complex DNN-based system like an autonomous vehicle is creating the system's specifications manually, against which the system's behavior can be checked. It is challenging to create detailed specifications for such a system as it essentially involves recreating the logic of a human driver. To avoid this issue, we leverage metamorphic relations [33] between the car behaviors across different synthetic images. The key insight is that even though it is hard to specify the correct behavior of a self-driving car for every transformed image, one can define relationships between the car's behaviors across certain types of transformations. For example, the autonomous car's steering angle should not change significantly for the same image under any lighting/weather conditions, blurring, or any affine transformations with small parameter values. Thus, if a DNN model infers a steering angle θ_o for an input seed image I_o and a steering angle θ_t for a new synthetic image I_t, which is generated by applying the transformation t on I_o, one may define a simple metamorphic relation where θ_o and θ_t are identical.

✓ IDEA

However, there is usually no single correct steering angle for a given image, i.e., a car can safely tolerate small variations. Therefore, there is a trade-off between defining the metamorphic relations very tightly, like the one described above (may result in a large number of false positives) and making the relations more permissive (may lead to many false negatives). In this paper, we strike a balance between these two extremes by using the metamorphic relations defined below.

To minimize false positives, we relax our metamorphic relations and allow variations within the error ranges of the original input images. We observe that the set of outputs predicted by a DNN model for the original images, say $\{\theta_{o1}, \theta_{o2}, ..., \theta_{on}\}$, in practice, result in a small but non-trivial number of errors *w.r.t.* their respective manual labels ($\{\hat{\theta}_1, \hat{\theta}_2, ..., \hat{\theta}_n\}$). Such errors are usually measured using Mean Squared Error (MSE), where $MSE_{orig} = \frac{1}{n}\sum_{i=1}^{n}(\hat{\theta}_i - \theta_{oi})^2$. Leveraging this property, we redefine a new metamorphic relation as:

$$(\hat{\theta}_i - \theta_{ti})^2 \leq \lambda\, MSE_{orig} \tag{2}$$

The above equation assumes that the errors produced by a model for the transformed images as input should be within a range of λ times the MSE produced by the original image set. Here, λ is a configurable parameter that allows us to strike a balance between the false positives and false negatives.

4 IMPLEMENTATION

Autonomous driving DNNs. We evaluate our techniques on three DNN models that won top positions in the Udacity self-driving challenge [15]: Rambo [13] (2^{nd} rank), Chauffeur [8] (3^{rd} rank), and Epoch [10] (6^{th} rank). We choose these three models as their implementations are based on the Keras framework [34] that our current prototype of DeepTest supports. The details of the DNN models and dataset are summarized in Table 3.

As shown in the right figure of Table 3, the steering angle is defined as the rotation degree between the heading direction of the vehicle (the vertical line) and the heading directions of the steering wheel axles (*i.e.*, usually front wheels). The negative steering angle indicates turning left while the positive values indicate turning left. The maximum steering angle of a car varies based on the hardware of different cars. The Udacity self-driving challenge dataset used in this paper has a maximum steering angle of +/- 25 degree [15]. The steering angle is then scaled by 1/25 so that the prediction should fall between -1 and 1.

Rambo model consists of three CNNs whose outputs are merged using a final layer [13]. Two of the CNNs are inspired by NVIDIA's

Model	Sub-Model	No. of Neurons	Reported MSE	Our MSE
Chauffeur	CNN LSTM	1427 513	0.06	0.06

Rambo	S1(CNN)
	S2(CNN)
	S3(CNN)
		1625
		3801
		13473
			0.06	0.05

Epoch	CNN	2500	0.08	0.10

† dataset HMB_3.bag [16]

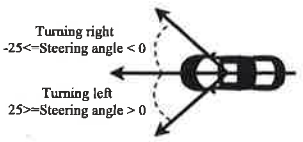


Table 3: (Left) Details of DNNs used to evaluate DeepTest.†(Right) The outputs of the DNNs are the steering angles for a self-driving car heading forward. The Udacity self-driving car has a maximum steering angle of +/- 25 degree.

self-driving car architecture [31], and the third CNN is based on comma.ai's steering model [9]. As opposed to other models that take individual images as input, Rambo takes the differences among three consecutive images as input. The model uses Keras [34] and Theano [79] frameworks.

Chauffeur model includes one CNN model for extracting features from the image and one LSTM model for predicting steering angle [8]. The input of the CNN model is an image while the input of the LSTM model is the concatenation of 100 features extracted by the CNN model from previous 100 consecutive images. Chauffeur uses Keras [34] and Tensorflow [24] frameworks.

Epoch model uses a single CNN. As the pre-trained model for Epoch is not publicly available, we train the model using the instructions provided by the authors [10]. We used the CH2_002 dataset [16] from the Udacity self-driving Challenge for training the epoch model. Epoch , similar to Chauffeur, uses Keras and Tensorflow frameworks.

Image transformations. In the experiments for RQ2 and RQ3, we leverage seven different types of simple image transformations: translation, scaling, horizontal shearing, rotation, contrast adjustment, brightness adjustment, and blurring. We use OpenCV to implement these transformations [7]. For RQ2 and RQ3 described in Section 5, we use 10 parameters for each transformation as shown in Table 4.

Table 4: Transformations and parameters used by DeepTest for generating synthetic images.

	Transformations	Parameters	Parameter ranges
	Translation	(t_x, t_y)	(10, 10) to (100, 100) step (10, 10)
	Scale	(s_x, s_y)	(1.5, 1.5) to (6, 6) step (0.5, 0.5)
	Shear	(s_x, s_y)	(−1.0, 0) to (−0.1, 0) step (0.1, 0)
	Rotation	q (degree)	3 to 30 with step 3
	Contrast	α (gain)	1.2 to 3.0 with step 0.2
	Brightness	β (bias)	10 to 100 with step 10
Blur	Averaging	kernel size	3 × 3, 4 × 4, 5 × 5, 6 × 6
	Gaussian	kernel size	3 × 3, 5 × 5, 7 × 7 , 3 × 3
	Median	aperture linear size	3, 5
	Bilateral Filter	diameter, sigmaColor, sigmaSpace	9, 75, 75

5 RESULTS

As DNN-based models are fundamentally different than traditional software, first, we check whether neuron coverage is a good metric to capture functional diversity of DNNs. In particular, we investigate whether neuron coverage changes with different input-output pairs of an autonomous car. An individual neuron's output goes through a sequence of linear and nonlinear operations before contributing to the final outputs of a DNN. Therefore, it is not very clear how much (if at all) individual neuron's activation will change the final output. We address this in our first research question.

Neuron . cov as a cov model ?

Table 5: Relation between neuron coverage and test output

Model	Sub-Model	Steering Angle	Steering Direction	
		Spearman Correlation	Wilcoxon Test	Effect size (Cohen's d)
Chauffeur	Overall CNN LSTM	-0.10 (***) 0.28 (***) -0.10 (***)	left (+ve) > right (-ve) (***) left (+ve) < right (-ve) (***) left (+ve) > right (-ve) (***)	negligible negligible negligible
Rambo	Overall S1 S2 S3	-0.11 (***) -0.19 (***) 0.10 (***) -0.11 (***)	left (+ve) < right (-ve) (***) left (+ve) < right (-ve) (***) not significant not significant	negligible large negligible negligible
Epoch	N/A	0.78 (***)	left (+ve) < right (-ve) (***)	small

*** indicates statistical significance with p-value $< 2.2 * 10^{-16}$

RQ1. Do different input-output pairs result in different neuron coverage?

For each input image we measure the neuron coverage (see Equation 1 in Section 3.1) of the underlying models and the corresponding output. As discussed in Section 4, corresponding to an input image, each model outputs a steering direction (left (+ve) / right (-ve)) and a steering angle as shown in Table 3 (right). We analyze the neuron coverage for both of these outputs separately.

Steering angle. As steering angle is a continuous variable, we check Spearman rank correlation [76] between neuron coverage and steering angle. This is a non-parametric measure to compute monotonic association between the two variables [44]. Correlation with positive statistical significance suggests that the steering angle increases with increasing neuron coverage and vice versa. Table 5 shows that Spearman correlations for all the models are statistically significant—while Chauffeur and Rambo models show an overall negative association, Epoch model shows a strong positive correlation. This result indicates that the neuron coverage changes with the changes in output steering angles, *i.e.* different neurons get activated for different outputs. Thus, in this setting, neuron coverage can be a good approximation for estimating the diversity of input-output pairs. Moreover, our finding that monotonic correlations between neuron coverage and steering angle also corroborate Goodfellow et al.'s hypothesis that, in practice, DNNs are often highly linear [40]. o.k.

Steering direction. To measure the association between neuron coverage and steering direction, we check whether the coverage varies between right and left steering direction. We use the Wilcoxon nonparametric test as the steering direction can only have two values (left and right). Our results confirm that neuron coverage varies with steering direction with statistical significance ($p < 2.2 * 10^{-16}$) for all the three overall models. Interestingly, for Rambo , only the Rambo-S1 sub-model shows statistically significant correlation but not Rambo-S2 and Rambo-S3. These results suggest that, unlike steering angle, some sub-models are more responsible than other for changing steering direction.

Overall, these results show that neuron coverage altogether varies significantly for different input-output pairs. Thus, a neuron-coverage-directed (NDG) testing strategy can help in finding corner cases.

? use NeuronCov as proxy for I/O diversity? questionable (human assessment 3D, images?)

Model	Sub-Model	No. of Neurons	Reported MSE	Our MSE
Chauffeur	CNN LSTM	1427 513	0.06	0.06

Rambo	S1(CNN)
	S2(CNN)
	S3(CNN)
		1625
		3801
		13473
			0.06	0.05

Epoch	CNN	2500	0.08	0.10

† dataset HMB_3.bag [16]

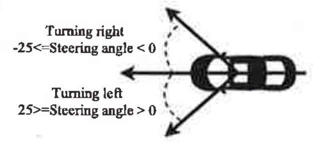


Table 3: (Left) Details of DNNs used to evaluate DeepTest.†(Right) The outputs of the DNNs are the steering angles for a self-driving car heading forward. The Udacity self-driving car has a maximum steering angle of +/- 25 degree.

self-driving car architecture [31], and the third CNN is based on comma.ai's steering model [9]. As opposed to other models that take individual images as input, Rambo takes the differences among three consecutive images as input. The model uses Keras [34] and Theano [79] frameworks.

Chauffeur model includes one CNN model for extracting features from the image and one LSTM model for predicting steering angle [8]. The input of the CNN model is an image while the input of the LSTM model is the concatenation of 100 features extracted by the CNN model from previous 100 consecutive images. Chauffeur uses Keras [34] and Tensorflow [24] frameworks.

Epoch model uses a single CNN. As the pre-trained model for Epoch is not publicly available, we train the model using the instructions provided by the authors [10]. We used the CH2_002 dataset [16] from the Udacity self-driving Challenge for training the epoch model. Epoch , similar to Chauffeur, uses Keras and Tensorflow frameworks.

Image transformations. In the experiments for RQ2 and RQ3, we leverage seven different types of simple image transformations: translation, scaling, horizontal shearing, rotation, contrast adjust-
ment, brightness adjustmen
implement these transforma
in Section 5, we use 10 parame
in Table 4.

**Table 4: Transformations anc
erating synthetic images.**

	Transformations	Paramete	[illegible]
	Translation	(t_x, t_y)	[illegible]
	Scale	(s_x, s_y)	[illegible]
	Shear	(s_x, s_y)	[illegible]
	Rotation	q (degree)	[illegible]
	Contrast	α (gain)	[illegible]
	Brightness	β (bias)	[illegible]
Blur	Averaging	kernel size	[illegible]
	Gaussian	kernel size	[illegible]
	Median	aperture linear size	3, 5
	Bilateral Filter	diameter, sigmaColor, sigmaSpace	9, 75, 75

Do different I/O pairs result in different neuron coverage?
Well if I_1 and I_2 only differ by rain, i.e. I_2 = rain(I_1), then $O_1 == O_2$ and Neuron cov constant?

5 RESULTS

As DNN-based models are fundamentally different than traditional software, first, we check whether neuron coverage is a good metric to capture functional diversity of DNNs. In particular, we investigate whether neuron coverage changes with different input-output pairs of an autonomous car. An individual neuron's output goes through a sequence of linear and nonlinear operations before contributing to the final outputs of a DNN. Therefore, it is not very clear how much (if at all) individual neuron's activation will change the final output. We address this in our first research question.

Neuron . cov as a cov model?

Table 5: Relation between neuron coverage and test output

Model	Sub-Model	Steering Angle	Steering Direction	
		Spearman Correlation	Wilcoxon Test	Effect size (Cohen's d)
Chauffeur	Overall CNN LSTM	-0.10 (***) 0.28 (***) -0.10 (***)	left (+ve) > right (-ve) (***) left (+ve) < right (-ve) (***) left (+ve) > right (-ve) (***)	negligible negligible negligible
Rambo	Overall S1 S2 S3	-0.11 (***) -0.19 (***) 0.10 (***) -0.11 (***)	left (+ve) < right (-ve) (***) left (+ve) < right (-ve) (***) not significant not significant	negligible large negligible negligible
Epoch	N/A	0.78 (***)	left (+ve) < right (-ve) (***)	small

*** indicates statistical significance with p-value $< 2.2 * 10^{-16}$

RQ1. Do different input-output pairs result in different neuron coverage?

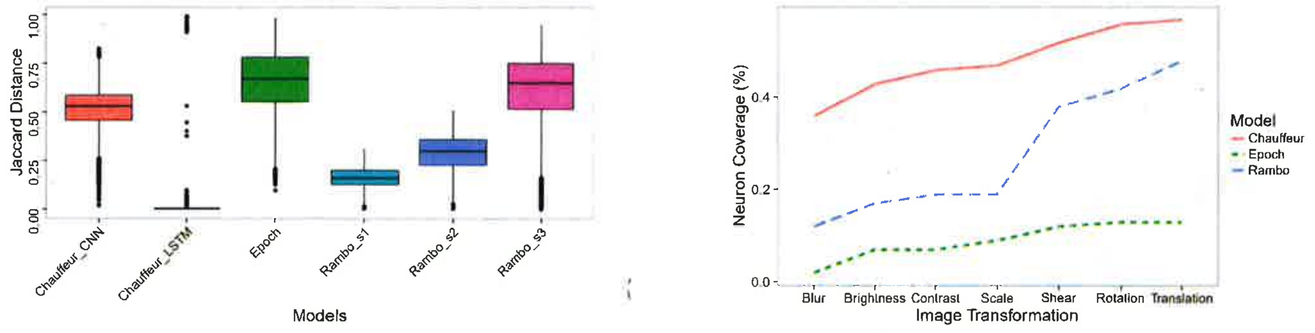
For each input image we measure the neuron coverage (see Equation 1 in Section 3.1) of the underlying models and the corresponding output. As discussed in Section 4, corresponding to an input image, each model outputs a steering direction (left (+ve) / right (-ve)) and a steering angle as shown in Table 3 (right). We analyze the neuron coverage for both of these outputs separately.

Steering angle. As steering angle is a continuous variable, we check Spearman rank correlation [76] between neuron coverage and steering angle. This is a non-parametric measure to compute monotonic association between the two variables [44]. Correlation
hat the steering angle
d vice versa. Table 5
odels are statistically
dels show an overall
ong positive correla-
erage changes with
fferent neurons get
setting, neuron cov-
ting the diversity of
: monotonic correla-
gle also corroborate
ce, DNNs are often

ation between neu-
whether the cover-
ection. We use the
direction can only
onfirm that neuron
coverage varies with steering direction with statistical significance ($p < 2.2 * 10^{-16}$) for all the three overall models. Interestingly, for Rambo , only the Rambo-S1 sub-model shows statistically significant correlation but not Rambo-S2 and Rambo-S3. These results suggest that, unlike steering angle, some sub-models are more responsible than other for changing steering direction.

Overall, these results show that neuron coverage altogether varies significantly for different input-output pairs. Thus, a neuron-coverage-directed (NDG) testing strategy can help in finding corner cases.

? use Neuron Cov as proxy for I/O diversity? questionable (human assessment & images?)



4.1 Difference in neuron coverage caused by different image transformations

4.2 Average cumulative neuron coverage per input image

Figure 4: Different image transformations activate significantly different neurons. In the top figure the median Jaccard distances for Chauffeur-CNN, Chauffeur-LSTM, Epoch, Rambo-S1, Rambo-S2, and Rambo-S3 models are 0.53, 0.002, 0.67, 0.12, 0.17, 0.30, and 0.65.

Result 1: *Neuron coverage is correlated with input-output diversity and can be used to systematic test generation.*

Next, we investigate whether synthetic images generated by applying different realistic image transformations (as described in Table 2) on existing input images can activate different neurons. Thus, we check:

RQ2. Do different realistic image transformations activate different neurons?

// seed images

We randomly pick 1,000 input images from the test set and transform each of them by using seven different transformations: blur, brightness, contrast, rotation, scale, shear, and translation. We also vary the parameters of each transformation and generate a total of 70,000 new synthetic images. We run all models with these synthetic images as input and record the neurons activated by each input.

We then compare the neurons activated by different synthetic images generated from the same image. Let us assume that two transformations T_1 and T_2, when applied to an original image I, activate two sets of neurons N_1 and N_2 respectively. We measure the dissimilarities between N_1 and N_2 by measuring their Jaccard distance: $1 - \frac{|N_1 \cap N_2|}{|N_1 \cup N_2|}$.

Figure 4.1 shows the result for all possible pair of transformations (*e.g.,* blur vs. rotation, rotation vs. transformation, *etc.*) for different models. These results indicate that for all models, except Chauffeur-LSTM , different transformations activate different neurons. As discussed in Section 2.1, LSTM is a particular type of RNN architecture that keeps states from previous inputs and hence increasing the neuron coverage of LSTM models with single transformations is much harder than other models. In this paper, we do not explore this problem any further and leave it as an interesting future work.

We further check how much a single transformation contributes in increasing the neuron coverage *w.r.t.* all other transformations for a given seed image. Thus, if an original image I undergoes seven discrete transformations: $T_1, T_2, ...T_7$, we compute the total number of neurons activated by T_1, $T_1 \cup T_2$, ..., $\bigcup_{i=1}^{7} T_i$. Figure 4.2 shows the cumulative effect of all the transformations on average neuron coverage per seed image. We see that the cumulative coverage increases with increasing number of transformations for all the models. In other words, all the transformations are contributing towards the overall neuron coverage.

We also compute the percentage change in neuron coverage per image transformation (N_T) *w.r.t.* to the corresponding seed image (N_O) as: $(N_T$-$N_O)/N_O$. Figure 5 shows the result. For all the studied models, the transformed images increase the neuron coverage significantly—Wilcoxon nonparametric test confirms the statistical significance. These results also show that different image transformations increase neuron coverage at different rates.

Result 2: *Different image transformations tend to activate different sets of neurons.* ? Should they ? *

Next, we mutate the seed images with different combinations of transformations (see Section 3). Since different image transformations activate different set of neurons, here we try to increase the neuron coverage by these transformed image inputs. To this end, we question:

RQ3. Can neuron coverage be further increased by combining different image transformations?

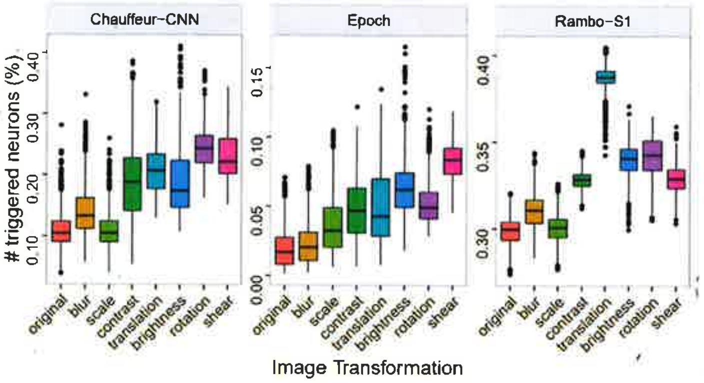
We perform this experiment by measuring neuron coverage in two different settings: (i) applying a set of transformations and (ii) combining transformations using *coverage-guided* search.

i) *Cumulative Transformations.* Since different seed images activate a different set of neurons (see RQ1), multiple seed images collectively achieve higher neuron coverage than a single one. Hence, we check whether the transformed images can still increase the neuron coverage collectively *w.r.t.* the cumulative baseline coverage of a set of seed images. In particular, we generate a total of 7,000 images from 100 seed images by applying 7 transformations and varying 10 parameters on 100 seed images. This results in a total of 7,000 test images. We then compare the cumulative neuron coverage of these synthetic images *w.r.t.* the baseline, which use the same 100 seed images for fair comparison. Table 6 shows the result. Across all the models (except Rambo-S3), the cumulative coverage increased significantly. Since the Rambo-S3 baseline already achieved 97% coverage, the transformed images only increase the coverage by $(13,080 - 13,008)/13,008 = 0.55\%$.

(How ?

ii) *Guided Transformations.* Finally, we check whether we can further increase the cumulative neuron coverage by using the coverage-guided search technique described in Algorithm 1. We generate 254, 221, and 864 images from 100 seed images for Chauffeur-CNN ,

* If the transformation only adds noise (blur/rain), should other neurons be activated & should the output change?



Median Increase in Neuron Coverage

Transformation	Chauffeur (CNN,LSTM)	Epoch	Rambo (S1,S2,S3)
Scale	(1.0,0.0) (0.67%,0%)	39.0** 93%	(2.0*,5.0*,32.0) (0.41%,1%,4%)
Brightness	(100.0**,1.0) (67%,0.2%)	113.0** 269%	(67.0**,104.0**,585.0*) (14%,24%,66%)
Contrast	(120.0**,1.0*) (80%,0.2%)	75.0** 179%	(47.0**,100.0**,159.0) (10%,23%,18%)
Blur	(41.0**,0.0) (28%,0%)	9.0* 21%	(18.0**,23.0**,269.5*) (4%,5%,31%)
Rotation	(199.0**,2.0*) (134%,0.39%)	81.0** 193%	(70.0**,13.0**,786.5*) (14%,3%,89%)
Translation	(147.0**,1.0*) (99%,0.2%)	65.0** 155%	(143.0**,167.0**,2315.5**) (29%,38%,263%)
Shear	(168.0**,1.0*) (113%,0.2%)	167.0** 398%	(48.0**,132.0**,1472.0**) (10%,30%,167%)

All numbers are statistically significant;
Numbers with * and ** have small and large Cohen's D effect.

Figure 5: Neuron coverage per seed image for individual image transformations *w.r.t.* baseline.

Table 6: Neuron coverage achieved by cumulative and guided transformations applied to 100 seed images.

Model	Baseline	Cumulative Transformation	Guided Generation	% increase of guided *w.r.t.* Baseline	Cumulative
Chauffeur-CNN	658 (46%)	1,065 (75%)	1,250 (88%)	90%	`17%
Epoch	621 (25%)	1034 (41%)	1,266 (51%)	104%	22%
Rambo-S1	710 (44%)	929 (57%)	1,043 (64%)	47%	12%
Rambo-S2	1,146 (30%)	2,210 (58%)	2,676 (70%)	134%	21%
Rambo-S3	13,008 (97%)	13,080 (97%)	13,150 (98%)	1.1%	0.5%

*

Epoch , and Rambo models respectively and measure their collective neuron coverage. As shown in Table 6, the guided transformations collectively achieve 88%, 51%, 64%, 70%, and 98% of total neurons for models Chauffeur-CNN , Epoch , Rambo-S1 , Rambo-S2 , and Rambo-S3 respectively, thus increasing the coverage up to 17% 22%, 12%, 21%, and 0.5% *w.r.t.* the unguided approach. This method also significantly achieves higher neuron coverage *w.r.t.* baseline cumulative coverage.

?

> **Result 3:** *By systematically combining different image transformations, neuron coverage can be improved by around 100% w.r.t. the coverage achieved by the original seed images.*

Next we check whether the synthetic images can trigger any erroneous behavior in the autonomous car DNNs and if we can detect those behaviors using metamorphic relations as described in Section 3.4. This leads to the following research question:

RQ4. Can we automatically detect erroneous behaviors using metamorphic relations?

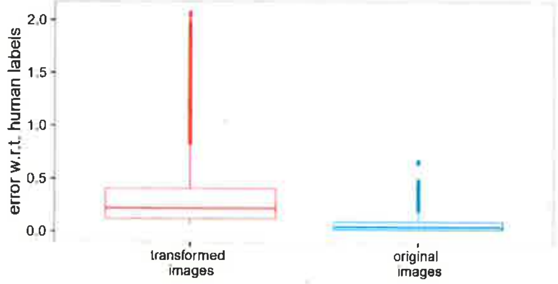


Figure 6: Deviations from the human labels for images that violate the metamorphic relation (see Equation 2) is higher compared to the deviations for original images. Thus, these synthetic images have a high chance to show erroneous behaviors.

Here we focus on the transformed images whose outputs violate the metamorphic relation defined in Equation 2. We call these images I_{err} and their corresponding original images as I_{org}. We compare the deviation between the outputs of I_{err} and I_{org} *w.r.t.* the corresponding human labels, as shown in Figure 6. The deviations produced for I_{err} are much larger than I_{org} (also confirmed by Wilcoxon test for statistical significance). In fact, mean squared error (MSE) for I_{err} is 0.41, while the MSE of the corresponding I_{org} is 0.035. Such differences also exist for other synthetic images that are generated by composite transformations including rain, fog, and those generated during the coverage-guided search. Thus, overall I_{err} has a higher potential to show buggy behavior.

However, for certain transformations (e.g., rotation), not all violations of the metamorphic relations can be considered buggy as the correct steering angle can vary widely based on the contents of the transformed image. For example, when an image is rotated by a large amount, say 30 degrees, it is nontrivial to automatically define its correct output behavior without knowing its contents. To reduce such false positives, we only report bugs for the transformations (e.g., small rotations, rain, fog, etc.) where the correct output should not deviate much from the labels of the corresponding seed images. Thus, we further use a filtration criteria as defined in Equation 3 to identify such transformations by checking whether the MSE of the synthetic images is close to that of the original images.

0, K.

$$| MSE_{(trans,param)} - MSE_{org} | \leq \epsilon \quad (3)$$

Thus, we only choose the transformations that obey Equation 3 for counting erroneous behaviors. Table 7 shows the number of such erroneous cases by varying two thresholds: ϵ and λ—a higher value of λ and lower value of ϵ makes the system report fewer bugs and vice versa. For example, with a λ of 5 and ϵ of 0.03, we report 330 violations for simple transformations. We do not enforce the filtration criteria for composite transformations. Rain and fog effects should produce same outputs as original images. Also, in guided search since multiple transformations produce the synthesized images, it is not possible to filter out a single transformation. Thus, for rain, fog, and guided search, we report 4448, 741, and 821 erroneous behavior respectively for $\lambda = 5$, across all three models.

Yes

* Why so high? Is it more 'dense' than the others?

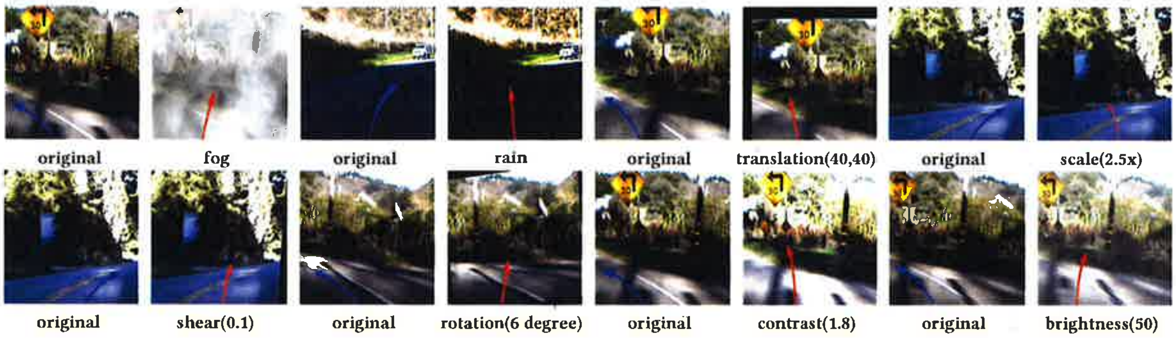


Figure 7: Sample images showing erroneous behaviors detected by DeepTest using synthetic images. For original images the arrows are marked in blue, while for the synthetic images they are marked in red. More such samples can be viewed at https://deeplearningtest.github.io/deepTest/.

Table 7: Number of erroneous behaviors reported by DeepTest across all tested models at different thresholds

λ	Simple Tranformation					Composite Transformation		
	ϵ (see Eqn. 3)					Fog	Rain	Guided Search
(see Eqn. 2)	0.01	0.02	0.03	0.04	0.05			
1	15666	18520	23391	24952	29649	9018	6133	1148
2	4066	5033	6778	7362	9259	6503	2650	1026
3	1396	1741	2414	2627	3376	5452	1483	930
4	501	642	965	1064	4884	4884	997	872
5	95	171	330	382	641	4448	741	820
6	49	85	185	210	359	4063	516	764
7	13	24	89	105	189	3732	287	721
8	3	5	34	45	103	3391	174	668
9	0	1	12	19	56	3070	111	637
10	0	0	3	5	23	2801	63	597

Table 8: Number of unique erroneous behaviors reported by DeepTest for different models with $\lambda = 5$ (see Eqn. 2)

Transformation	Chauffeur	Epoch	Rambo
Simple Transformation			
Blur	3	27	11
Brightness	97	32	15
Contrast	31	12	-
Rotation	-	13	-
Scale	-	10	-
Shear	-	-	23
Translation	21	35	-
Composite Transformation			
Rain	650	64	27
Fog	201	135	4112
Guided	89	65	666

Table 8 further elaborates the result for different models for $\lambda = 5$ and $\epsilon = 0.03$, as highlighted in Table 7. Interestingly, some models are more prone to erroneous behaviors for some transformations than others. For example, Rambo produces 23 erroneous cases for shear, while the other two models do not show any such cases. Similarly, DeepTest finds 650 instances of erroneous behavior in Chauffeur for rain but only 64 and 27 for Epoch and Rambo respectively. In total, DeepTest detects 6339 erroneous behaviors across all three models. Figure 7 further shows some of the erroneous behaviors that are detected by DeepTest under different transformations that can lead to potentially fatal situations.

We also manually checked the bugs reported in Table 8 and report the false positives in Figure 8. It also shows two synthetic images (the corresponding original images) where DeepTest incorrectly reports erroneous behaviors while the model's output is indeed safe. Although such manual investigation is, by definition, subjective and approximate, all the authors have reviewed the images and agreed on the false positives.

Model	Simple Transformation	Guided	Rain	Fog	Total
Epoch	14	0	0	0	14
Chauffeur	5	3	12	6	26
Rambo	8	43	11	28	90
Total	27	46	23	34	130

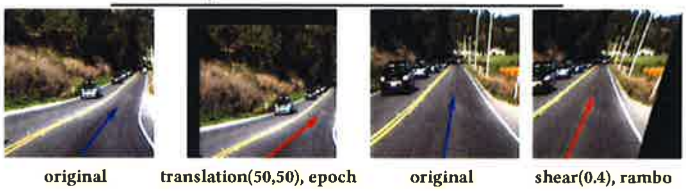


Figure 8: Sample false positives produced by DeepTest for $\lambda = 5$, $\epsilon = 0.03$

> **Result 4:** *With neuron coverage guided synthesized images, DeepTest successfully detects more than 1,000 erroneous behavior as predicted by the three models with low false positives.*

RQ5. Can retraining DNNs with synthetic images improve accuracy?

Table 9: Improvement in MSE after retraining of Epoch model with synthetic tests generated by DeepTest

Test set	Original MSE	Retrained MSE
original images	0.10	0.09
with fog	0.18	0.10
with rain	0.13	0.07

?
(coverage?)

Here we check whether retraining the DNNs with some of the synthetic images generated by DeepTest helps in making the DNNs more robust.We used the images from HMB_3.bag [16] and created their synthetic versions by adding the rain and fog effects. We retrained the Epoch model with randomly sampled 66% of these synthetic inputs along with the original training data. We evaluated both the original and the retrained model on the rest 34% of the synthetic images and their original versions. In all cases, the accuracy of the retrained model improved significantly over the original model as shown in Table 9.

> **Result 5:** *Accuracy of a DNN can be improved up to 46% by retraining the DNN with synthetic data generated by DeepTest.*

rain/fog are easy because the output should be the same!
What about coverage?

6 THREATS TO VALIDITY

DeepTest generates realistic synthetic images by applying different image transformations on the seed images. However, these transformations are not designed to be exhaustive and therefore may not cover all realistic cases.

While our transformations like rain and fog effects are designed to be realistic, the generated pictures may not be exactly reproducible in reality due to a large number of unpredictable factors, e.g., the position of the sun, the angle and size of the rain drops, etc. However, as the image processing techniques become more sophisticated, the generated pictures will get closer to reality.

A complete DNN model for driving an autonomous vehicle must also handle braking and acceleration besides the steering angle. We restricted ourselves to only test the accuracy of the steering angle as our tested models do not support braking and acceleration yet. However, our techniques should be readily applicable to testing those outputs too assuming that the models support them.

7 RELATED WORK

Testing of driver assistance systems. Abdessalem et al. proposed a technique for testing Advanced Driver Assistance Systems (ADAS) in autonomous cars that show warnings to the drivers if it detects pedestrians in positions with low driver visibility [25]. They use multi-objective meta heuristic search algorithms to generate tests that simultaneously focus on the most critical behaviors of the system and the environment as decided by the domain experts (*e.g.*, moving pedestrians in the dark).

The key differences between this work and ours are threefold: (i) We focus on testing the image recognition and steering logic in the autonomous car DNNs while their technique tested ADAS system's warning logic based on preprocessed sensor inputs; (ii) Their blackbox technique depends on manually selected critical scenarios while our gray-box technique looks inside the DNN model and systematically maximize neuron coverage. The trade-off is that their technique can, in theory, work for arbitrary implementations while our technique is tailored for DNNs; and (iii) We leverage metamorphic relations for creating a test oracle while they depend on manual specifications for detecting faulty behavior.

Testing and verification of machine learning. Traditional practices in evaluating machine learning systems primarily measure their accuracy on randomly drawn test inputs from manually labeled datasets and ad hoc simulations [11, 20, 82]. However, without the knowledge of the model's internals, such blackbox testing paradigms are not able to find different corner-cases that may induce unexpected behaviors [26, 70].

Pei *et al.* [70] proposed DeepXplore, a whitebox differential testing algorithm for systematically finding inputs that can trigger inconsistencies between multiple DNNs. They introduced neuron coverage as a systematic metric for measuring how much of the internal logic of a DNNs have been tested. By contrast, our graybox methods use neuron coverage for guided test generation in a single DNN and leverage metamorphic relations to identify erroneous behaviors without requiring multiple DNNs.

Another recent line of work has explored the possibility of verifying DNNs against different safety properties [48, 51, 71]. However, none of these techniques can verify a rich set of properties for real-world-sized DNNs. By contrast, our techniques can systematically test state-of-the-art DNNs for safety-critical erroneous behaviors but do not provide any theoretical guarantees.

don't scale

yes

Adversarial machine learning. A large number of projects successfully attacked machine learning models at test time by forcing it to make unexpected mistakes. More specifically, these attacks focus on finding inputs that, when changed minimally from their original versions, get classified differently by the machine learning classifiers. These types of attacks are known to affect a broad spectrum of tasks such as image recognition [37, 40, 52, 55, 62, 63, 65, 66, 78], face detection/verification [75, 81], malware detection [28, 42, 54, 85], and text analysis [59, 67]. Several prior works have explored defenses against these attacks with different effectiveness [29, 32, 35, 38, 41, 43, 48, 57, 64, 68, 74, 77, 80, 84, 86].

In summary, this line of work tries to find a particular class of erroneous behaviors, i.e., forcing incorrect prediction by adding a minimum amount of noise to a given input. By contrast, we systematically test a given DNN by maximizing neuron coverage and find a diverse set of corner-case behaviors. Moreover, we specifically focus on finding realistic conditions that can occur in practice.

Test amplification. There is a large body of work on test case generation and amplification techniques for traditional software that automatically generate test cases from some seed inputs and increase code coverage. Instead of summarizing them individually here, we refer the interested readers to the surveys by Anand et al. [27], McMinn et al. [56], and Pasareanu et al. [69]. Unlike these approaches, DeepTest is designed to operate on DNNs.

Metamorphic testing. Metamorphic testing [33, 87] is a way of creating test oracles in settings where manual specifications are not available. Metamorphic testing identifies buggy behavior by detecting violations of domain-specific metamorphic relations that are defined across outputs from multiple executions of the test program with different inputs. For example, a sample metamorphic property for program p adding two inputs a and b can be $p(a, b) = p(a, 0) + p(b, 0)$. Metamorphic testing has been used in the past for testing both supervised and unsupervised machine learning classifiers [60, 83]. By contrast, we define new metamorphic relations in the domain of autonomous cars which, unlike the classifiers tested before, produce a continuous steering angle, i.e., it is a regression task. unclear

8 CONCLUSION

In this paper, we proposed and evaluated DeepTest, a tool for automated testing of DNN-driven autonomous cars. DeepTest maximizes the neuron coverage of a DNN using synthetic test images generated by applying different realistic transformations on a set of seed images. We use domain-specific metamorphic relations to find erroneous behaviors of the DNN without detailed specification. DeepTest can be easily adapted to test other DNN-based systems by customizing the transformations and metamorphic relations. We believe DeepTest is an important first step towards building robust DNN-based systems.

9 ACKNOWLEDGEMENTS

We would like to thank Yoav Hollander and the anonymous reviewers for their helpful feedback. This work was supported in part by NSF grants CNS-16-18771, CNS-16-17670, and CNS-15-64055; ONR grant N00014-17-1-2010; and a Google Faculty Fellowship.

REFERENCES

[1] 2013. Add Dramatic Rain to a Photo in Photoshop. https://design.tutsplus.com/tutorials/add-dramatic-rain-to-a-photo-in-photoshop--psd-29536. (2013).

[2] 2013. How to create mist: Photoshop effects for atmospheric landscapes. http://www.techradar.com/how-to/photography-video-capture/cameras/how-to-create-mist-photoshop-effects-for-atmospheric-landscapes-1320997. (2013).

[3] 2014. *The OpenCV Reference Manual* (2.4.9.0 ed.).

[4] 2014. This Is How Bad Self-Driving Cars Suck In The Rain. http://jalopnik.com/this-is-how-bad-self-driving-cars-suck-in-the-rain-1666268433. (2014).

[5] 2015. Affine Transformation. https://www.mathworks.com/discovery/affine-transformation.html. (2015).

[6] 2015. Affine Transformations. http://docs.opencv.org/3.1.0/d4/d61/tutorial_warp_affine.html. (2015).

[7] 2015. Open Source Computer Vision Library. https://github.com/itseez/opencv. (2015).

[8] 2016. Chauffeur model. https://github.com/udacity/self-driving-car/tree/master/steering-models/community-models/chauffeur. (2016).

[9] 2016. comma.ai's steering model. https://github.com/commaai/research/blob/master/train_steering_model.py. (2016).

[10] 2016. Epoch model. https://github.com/udacity/self-driving-car/tree/master/steering-models/community-models/cg23. (2016).

[11] 2016. Google Auto Waymo Disengagement Report for Autonomous Driving. https://www.dmv.ca.gov/portal/wcm/connect/946b3502-c959-4e3b-b119-91319c27788f/GoogleAutoWaymo_disengage_report_2016.pdf?MOD=AJPERES. (2016).

[12] 2016. Google's Self-Driving Car Caused Its First Crash. https://www.wired.com/2016/02/googles-self-driving-car-may-caused-first-crash/. (2016).

[13] 2016. Rambo model. https://github.com/udacity/self-driving-car/tree/master/steering-models/community-models/rambo. (2016).

[14] 2016. Tesla Autopilot. https://www.tesla.com/autopilot. (2016).

[15] 2016. Udacity self driving car challenge 2. https://github.com/udacity/self-driving-car/tree/master/challenges/challenge-2. (2016).

[16] 2016. Udacity self driving car challenge 2 dataset. https://github.com/udacity/self-driving-car/tree/master/datasets/CH2. (2016).

[17] 2016. Who's responsible when an autonomous car crashes? http://money.cnn.com/2016/07/07/technology/tesla-liability-risk/index.html. (2016).

[18] 2017. Autonomous Vehicles Enacted Legislation. http://www.ncsl.org/research/transportation/autonomous-vehicles-self-driving-vehicles-enacted-legislation.aspx. (2017).

[19] 2017. Baidu Apollo. https://github.com/ApolloAuto/apollo. (2017).

[20] 2017. Inside Waymo's Secret World for Training Self-Driving Cars. https://www.theatlantic.com/technology/archive/2017/08/inside-waymos-secret-testing-and-simulation-facilities/537648/. (2017).

[21] 2017. The Numbers Don't Lie: Self-Driving Cars Are Getting Good. https://www.wired.com/2017/02/california-dmv-autonomous-car-disengagement. (2017).

[22] 2017. Software 2.0. https://medium.com/@karpathy/software-2-0-a64152b37c35. (2017).

[23] 2017. Tesla's Self-Driving System Cleared in Deadly Crash. https://www.nytimes.com/2017/01/19/business/tesla-model-s-autopilot-fatal-crash.html. (2017).

[24] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016).

[25] Raja Ben Abdessalem, Shiva Nejati, Lionel C Briand, and Thomas Stifter. 2016. Testing advanced driver assistance systems using multi-objective search and neural networks. In *Automated Software Engineering (ASE), 2016 31st IEEE/ACM International Conference on*. IEEE, 63–74.

[26] an Goodfellow and Nicolas Papernot. 2017. The challenge of verification and testing of machine learning. http://www.cleverhans.io/security/privacy/ml/2017/06/14/verification.html. (2017).

[27] Saswat Anand, Edmund K Burke, Tsong Yueh Chen, John Clark, Myra B Cohen, Wolfgang Grieskamp, Mark Harman, Mary Jean Harrold, Phil Mcminn, Antonia Bertolino, et al. 2013. An orchestrated survey of methodologies for automated software test case generation. *Journal of Systems and Software* 86, 8 (2013), 1978–2001.

✱ ✱ ✱

[28] Hyrum Anderson. 2017. Evading Next-Gen AV using A.I. https://www.defcon.org/html/defcon-25/dc-25-index.html. (2017).

[29] Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya Nori, and Antonio Criminisi. 2016. Measuring neural net robustness with constraints. In *Advances in Neural Information Processing Systems*. 2613–2621.

[30] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. 2007. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*. 153–160.

[31] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* (2016).

[32] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 39–57.

Δ Metamorphic testing

[33] Tsong Y Chen, Shing C Cheung, and Shiu Ming Yiu. 1998. *Metamorphic testing: a new approach for generating next test cases*. Technical Report. Technical Report HKUST-CS98-01, Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong.

[34] François Chollet et al. 2015. Keras. https://github.com/fchollet/keras. (2015).

[35] Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. 2017. Parseval networks: Improving robustness to adversarial examples. In *International Conference on Machine Learning*. 854–863.

[36] California DMV. 2016. Autonomous Vehicle Disengagement Reports. https://www.dmv.ca.gov/portal/dmv/detail/vr/autonomous/disengagement_report_2016. (2016).

[37] Ivan Evtimov, Kevin Eykholt, Earlence Fernandes, Tadayoshi Kohno, Bo Li, Atul Prakash, Amir Rahmati, and Dawn Song. 2017. Robust Physical-World Attacks on Machine Learning Models. *arXiv preprint arXiv:1707.08945* (2017).

[38] Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. 2017. Detecting Adversarial Samples from Artifacts. *arXiv preprint arXiv:1703.00410* (2017).

[39] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. http://www.deeplearningbook.org Book in preparation for MIT Press.

[40] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations (ICLR)*.

[41] Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick McDaniel. 2017. On the (statistical) detection of adversarial examples. *arXiv preprint arXiv:1702.06280* (2017).

[42] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick D. McDaniel. 2017. Adversarial Perturbations Against Deep Neural Networks for Malware Classification. In *Proceedings of the 2017 European Symposium on Research in Computer Security*.

[43] Shixiang Gu and Luca Rigazio. 2015. Towards deep neural network architectures robust to adversarial examples. In *International Conference on Learning Representations (ICLR)*.

[44] Jan Hauke and Tomasz Kossowski. 2011. Comparison of values of Pearson's and Spearman's correlation coefficients on the same sets of data. *Quaestiones geographicae* 30, 2 (2011), 87.

[45] Samer Hijazi, Rishi Kumar, and Chris Rowen. 2015. *Using convolutional neural networks for image recognition*. Technical Report. Tech. Rep., 2015.[Online]. Available: http://ip. cadence. com/uploads/901/cnn-wp-pdf.

[46] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. 2001. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. (2001).

[47] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[48] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. 2017. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification*. Springer, 3–29.

[49] L. C. Jain and L. R. Medsker. 1999. *Recurrent Neural Networks: Design and Applications* (1st ed.). CRC Press, Inc., Boca Raton, FL, USA.

[50] Andrej Karpathy. [n. d.]. Convolutional neural networks. http://cs231n.github.io/convolutional-networks/. ([n. d.]).

[51] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2017. *Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks*. Springer International Publishing, Cham, 97–117.

[52] Jernej Kos, Ian Fischer, and Dawn Song. 2017. Adversarial examples for generative models. *arXiv preprint arXiv:1702.06832* (2017).

[53] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[54] Pavel Laskov et al. 2014. Practical evasion of a learning-based classifier: A case study. In *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE, 197–211.

[55] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Xiaodong Song. 2017. Delving into Transferable Adversarial Examples and Black-box Attacks. In *International Conference on Learning Representations (ICLR)*.

[56] Phil McMinn. 2004. Search-based software test data generation: a survey. *Software testing, Verification and reliability* 14, 2 (2004), 105–156.

[57] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. 2017. On detecting adversarial perturbations. In *International Conference on Learning Representations (ICLR)*.

[58] Thomas M. Mitchell. 1997. *Machine Learning* (1 ed.). McGraw-Hill, Inc., New York, NY, USA.

[59] Takeru Miyato, Andrew M Dai, and Ian Goodfellow. 2016. Adversarial Training Methods for Semi-Supervised Text Classification. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

[60] Christian Murphy, Gail E Kaiser, Lifeng Hu, and Leon Wu. 2008. Properties of Machine Learning Applications for Use in Metamorphic Testing.. In *SEKE*, Vol. 8. 867–872.

[61] Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference*

O SWTestGen surveys

End2end learning

on machine learning (ICML-10)*. 807–814.

[62] Nina Narodytska and Shiva Prasad Kasiviswanathan. 2016. Simple black-box adversarial perturbations for deep networks. In *Workshop on Adversarial Training, NIPS 2016.*

[63] Anh Nguyen, Jason Yosinski, and Jeff Clune. 2015. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 427–436.

[64] Nicolas Papernot and Patrick McDaniel. 2017. Extending Defensive Distillation. *arXiv preprint arXiv:1705.05264* (2017).

[65] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. 2017. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ACM, 506–519.

[66] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 372–387.

[67] Nicolas Papernot, Patrick McDaniel, Ananthram Swami, and Richard Harang. 2016. Crafting adversarial input sequences for recurrent neural networks. In *Military Communications Conference, MILCOM 2016-2016 IEEE*. IEEE, 49–54.

[68] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2016. Distillation as a defense to adversarial perturbations against deep neural networks. In *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 582–597.

[69] Corina S Păsăreanu and Willem Visser. 2009. A survey of new trends in symbolic execution for software testing and analysis. *International Journal on Software Tools for Technology Transfer (STTT)* 11, 4 (2009), 339–353.

[70] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. *arXiv preprint arXiv:1705.06640* (2017). ++ Neuron Coverage

[71] Luca Pulina and Armando Tacchella. 2010. An abstraction-refinement approach to verification of artificial neural networks. In *Computer Aided Verification*. Springer, 243–257.

[72] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1988. Learning representations by back-propagating errors. *Cognitive modeling* 5, 3 (1988), 1.

[73] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, and Michael Young. 2014. Machine Learning: The High Interest Credit Card of Technical Debt.

[74] Uri Shaham, Yutaro Yamada, and Sahand Negahban. 2015. Understanding adversarial training: Increasing local stability of neural nets through robust optimization. *arXiv preprint arXiv:1511.05432* (2015).

[75] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K Reiter. 2016. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1528–1540.

[76] Charles Spearman. 1904. The proof and measurement of association between two things. *The American journal of psychology* 15, 1 (1904), 72–101.

[77] Jacob Steinhardt, Pang Wei Koh, and Percy Liang. 2017. Certified Defenses for Data Poisoning Attacks. *arXiv preprint arXiv:1706.03691* (2017).

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

[79] Theano Development Team. 2016. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints* abs/1605.02688 (May 2016). http://arxiv.org/abs/1605.02688

[80] Gang Wang, Tianyi Wang, Haitao Zheng, and Ben Y Zhao. 2014. Man vs. Machine: Practical Adversarial Detection of Malicious Crowdsourcing Workers.. In *USENIX Security Symposium*. 239–254.

[81] Michael J Wilber, Vitaly Shmatikov, and Serge Belongie. 2016. Can we still avoid automatic face detection?. In *Applications of Computer Vision (WACV), 2016 IEEE Winter Conference on*. IEEE, 1–9.

[82] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. 2016. *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann.

[83] Xiaoyuan Xie, Joshua Ho, Christian Murphy, Gail Kaiser, Baowen Xu, and Tsong Yueh Chen. 2009. Application of metamorphic testing to supervised classifiers. In *Quality Software, 2009. QSIC'09. 9th International Conference on*. IEEE, 135–144.

[84] Weilin Xu, David Evans, and Yanjun Qi. 2017. Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks. *arXiv preprint arXiv:1704.01155* (2017).

[85] Weilin Xu, Yanjun Qi, and David Evans. 2016. Automatically evading classifiers. In *Proceedings of the 2016 Network and Distributed Systems Symposium.*

[86] Stephan Zheng, Yang Song, Thomas Leung, and Ian Goodfellow. 2016. Improving the robustness of deep neural networks via stability training. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4480–4488.

[87] Zhi Quan Zhou, DH Huang, TH Tse, Zongyuan Yang, Haitao Huang, and TY Chen. 2004. Metamorphic testing and its applications. In *Proceedings of the 8th International Symposium on Future Software Technology (ISFST 2004)*. 346–351.

↗ metamorphic testing △