Detect and Repair Bugs/Errors in DNN-based Software

Thesis proposal

Yuchi Tian
Department of Computer Science
Columbia University
yuchi@cs.columbia.edu

December 11, 2020

Abstract

In this thesis proposal, we try to solve several important problems in SE (software engineering) for AI (artificial intelligence). As DNN(deep neural networks) becomes popular, more and more DNN-based software appears. It is essential to propose new approaches to test and repair them because methodologies developed for traditional software cannot be directly applied in DNN-based software. We have proposed novel approaches to enable developers to test, detect and repair bugs/errors for DNN based software to ensure functionality, security as well as fairness. This thesis is comprised of three primary publications: 1) DeepTest, a systematic approach leveraging realistic transformations to automatically generate test inputs and detect erroneous behaviours in DNN-based self-driving car, which was published in ICSE 2018[88]; 2) DeepInspect, leveraging NAPVD(neuron activation probability vector distance) to automatically detect confusion and bias errors in DNN-based image classifiers, which was published in ICSE 2020[89]; and 3) Proposing differentiable distance metric to repair confusion errors, which was published in FSE 2020[87]. This thesis focuses on solving the important issues in software engineering for the DNN-based software with respect to functionality, security and fairness.

Contents

1 Introduction

Nowadays, DNNs(deep neural networks) become more and more popular. Large numbers of DNN-based software have been developed and deployed in many areas such as computer vision, natural language processing, medical image analysis, bioinformatics, self-driving cars, machine translation, bugs detection and repair and so on.[79]. DNN-based software have been considered as software 2.0[15]. It is essential to propose new approaches to ensure the functionality, security as well as fairness because methodologies developed for traditional software cannot be directly applied in DNN-based software.

This thesis proposal explores and solves several important problems in this new SE for AI area, the intersection area between software engineering and artificial intelligence. We propose a systematic tool to enable developers to automatically test DNN-based self-driving car and identify erroneous behaviours which can potentially lead to fatal crashes. We also proposed a testing approach to automatically detect confusion and bias errors in DNN-based image classifiers. Then we proposed approaches to enable developers to repair these confusion and bias errors.

This thesis proposal discusses three primary works: 1)We proposed DeepTest, a systematic testing tool to automatically detect erroneous behaviours, which can potentially lead to fatal crashes. It leverages realistic transformations including different weather conditions to automatically generate test inputs, which maximize the neuron coverage. This work is published in ICSE 2018[88]. 2) We proposed DeepInspect to automatically detect confusion and bias errors for DNN-based image classifiers. It leverages NAPVD(neuron activation probability vector distance) to identify confusion and bias errors in DNN-based image classifiers. We show that the smaller the distance is the more confused the model is between two classes. This work is published in ICSE 2020[89]. 3) We proposed a differentiable distance metric correlated with confusion errors and a repairing approach by increasing the distance between two classes during retraining a model to reduce confusion errors. This work is published in ESEC/FSE 2020[87].

1.1 DNN Testing Related Work

Testing & Verification of DNNs.
Inspired from coverage metric in traditional software such as branch coverage, path coverage, MC/DC coverage metric and etc., coverage metrics such as neuron coverage[74], k-multisection coverage[55], neuron boundary coverage[55] and sign-sign coverage[83] have been proposed for DNN based software to evaluate the test inputs quality or to indicate whether they are well tested or not. Then different white-box and black-box testing approaches based on these new metrics have been proposed [74, 88, 100, 84, 38]. Sun *et al.* [84] presented a concolic testing approach for DNNs called DeepConcolic. They showed that their concolic testing approach can effectively increase coverage and find adversarial examples. Odena and Goodfellow proposed TensorFuzz[66], a general tool that combines coverage-guided fuzzing with property-based testing to generate cases that violate a user-specified objective. It has applications like finding numerical errors in trained neural networks, exposing disagreements between neural networks and their quantized versions, surfacing broken loss functions in popular GitHub repositories, and making performance improvements to TensorFlow. There are also efforts to verify DNNs [75, 44, 43, 91] against adversarial attacks. However, most of the verification efforts are limited to small DNNs and pixel-level properties.

Adversarial Deep Learning.
DNNs are vulnerable to well-crafted inputs called adversarial examples, where the discrepancies are imperceptible to a human but can easily fool deep neural network models [95, 53, 76, 67, 32, 35, 46, 64, 65, 70, 71, 85, 42, 48]. There are also studies in defending against adversarial attacks [19, 25, 33, 36, 37, 59, 68, 72, 81, 93, 103, 41, 56]. The generated adversarial examples are mostly used for evaluating how well the pre-trained DNNs are robust against malicious attack, which is different from evaluating the functionality in realistic scenarios since most of these works leveraged pixel based perturbation which may not exist in real scenarios. Our first work leveraged realistic transformations including rain or fog condition to identify thousands of erroneous behaviours in self-driving cars, which can potentially lead to fatal crashes in reality.

Interpreting DNNs.
There has been much research on model interpretability and visualization [52, 101, 80, 61, 20, 29]. A comprehensive study is presented by Lipton [52]. Dong *et al.* [29] observed that instead of learning the semantic features of whole objects, neurons tend to react to different parts of the objects in a recurrent manner. In our second work, our probabilistic way of looking at neuron activation per class aims to capture holistic behavior of an entire class instead of an individual object so diverse features of class members can be captured.

1.2 Software Testing Related Work

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. [17], McMinn et al. [57], and Pasareanu et al. [73]. Inspired from test amplification for traditional software, one of the contributions in our first work DeepTest is to automatically generate test inputs for DNN based software to maximize the neuron coverage.

Metamorphic testing.
Metamorphic testing [26, 104] 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 [62, 92]. In our first work DeepTest, 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.

1.3 Bias/Fairness Related Work

Evaluating Models' Bias/Fairness. Evaluating the bias and fairness of a system is important both from a theoretical and a practical perspective [54, 99, 98, 23]. Related studies first define a

fairness criteria and then try to optimize the original objective while satisfying the fairness criteria [31, 39, 18, 58, 30, 50]. These properties are defined either at individual [31, 49, 45] or group levels [24, 39, 97].

Galhotra *et al.* [34] first applied the notion of software testing to evaluating software fairness. They mutate the sensitive features of the inputs and check whether the output changes. One major problem with their proposed method, Themis, is that it assumes the model takes into account sensitive attribute(s) during training and inference. This assumption is not realistic since most existing fairness-aware models drop input-sensitive feature(s). Besides, Themis will not work on image classification, where the sensitive attribute (e.g., gender, race) is a visual concept that cannot be flipped easily.

In our second work DeepInspect, we proposed a white-box approach to measure the bias learned by the model during training. Our testing method does not require the model to take into account any sensitive feature(s). We propose a new fairness notion for the setting of multi-object classification, *average confusion disparity*, and a proxy, *average bias*, to measure for any deep learning model even when only unlabeled testing data is provided. In addition, our method tries to provide an explanation behind the discrimination. A complementary approach by Papernot *et al.* [69] shows such explainability behind model bias in a single classification setting.

2 DeepTest

2.1 Introduction

In this work, we leveraged the idea from traditional software and designed a systematic testing tool, DeepTest, to test DNN-based self-driving cars. Many companies including Tesla, GM, Ford, BMW, and Waymo/Google are building and actively testing these cars[14, 28]. However, 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. 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 [7, 13] and therefore miss numerous corner cases.

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. We address these issues and design a systematic testing methodology for automatically detecting erroneous behaviors of DNN-based software of self-driving cars.

2.2 Background

2.2.1 Deep Learning for Autonomous Driving

The key component of an autonomous vehicle is the perception module controlled by the underlying Deep Neural Network (DNN) [12, 9]. 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 current conditions as shown in Figure 1. In this work, we focus on the camera input and the steering angle output.

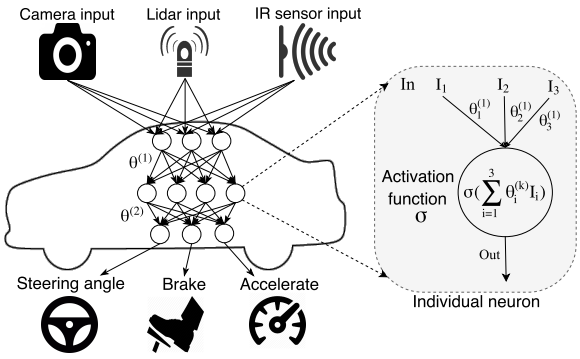


Figure 1: A simple autonomous car DNN that takes inputs from camera, light detection and ranging sensor (LiDAR), and IR (infrared) sensor, and outputs steering angle, braking decision, and acceleration decision. The DNN shown here essentially models the function $\sigma(\boldsymbol{\theta^{(2)}} \cdot \sigma(\boldsymbol{\theta^{(1)}} \cdot \boldsymbol{x}))$ 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.

A typical feed-forward DNN is composed of multiple processing *layers* stacked together to extract different representations of the input [21]. 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 ($\boldsymbol{\theta}$s in Figure 1). Each neuron applies a nonlinear *activation function* on its inputs and sends the output to the subsequent neurons as shown in Figure 1. Popular activation functions include ReLU (Rectified Linear Unit) [63], sigmoid [60], 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 [77]. 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 1 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 [22].

2.3 Methodology

In order for DeepTest to work, we need to solve the following four problems. (i) How to systematically explore the input-output spaces of DNN-based an autonomous car? (ii) How to synthesize

realistic inputs to automate such exploration? (iii) How to optimize the exploration process? (iv) How to automatically create a test oracle that can detect erroneous behaviors without detailed manual specifications? Each of these questions is addressed below in a section.

2.3.1 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 [74] 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).

Neuron coverage was originally proposed by Pei *et al.* for guided differential testing of multiple similar DNNs [74]. 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 2.5.

2.3.2 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). We compose multiple filters provided by Adobe Photoshop on the input images to simulate realistic fog and rain effects [1, 2]. Our experimental results demonstrate that all of these transformations increase neuron coverage significantly for all of the tested DNNs.

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 = φ
while S is not empty do
    img = S.pop()
    Tqueue = φ
    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
```

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.

2.3.4 Test Oracle using Metamorphic Relations

We leverage metamorphic relations [26] 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.

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). 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.

2.4 Implementation

Autonomous driving DNNs. We evaluate our techniques on three DNN models that won top positions in the Udacity self-driving challenge [10]: Rambo [8] (2^{nd} rank), Chauffeur [4] (3^{rd} rank), and Epoch [6] (6^{th} rank). We choose these three models as their implementations are based on the Keras framework [27] that our current prototype of DeepTest supports. The details of the DNN models and dataset are summarized in Table 1.

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 [11]

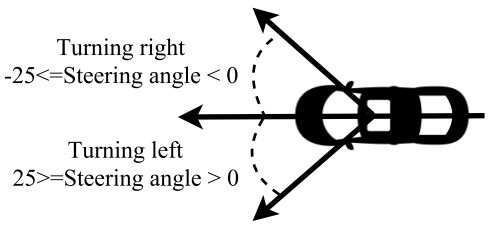


Table 1: (Left) Details of DNNs used to evaluate DeepTest.†(Right) The outputs of the DNNs are the steering angles for a self-driving car heading forward. The Udacity self-driving car has a maximum steering angle of +/- 25 degree.

As shown in the right figure of Table 1, 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 [10]. 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 [8]. Two of the CNNs are inspired by NVIDIA's self-driving car architecture [22], and the third CNN is based on comma.ai's steering model [5]. 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 [27] and Theano [86] frameworks.

Chauffeur model includes one CNN model for extracting features from the image and one LSTM model for predicting steering angle [4]. 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 [27] and Tensorflow [16] 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 [6]. We used the CH2_002 dataset [11] 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, 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 [3]. We use 10 parameters for each transformation as shown in Table 2.

Table 2: 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	$3 \times 3, 4 \times 4, 5 \times 5, 6 \times 6$
	Gaussian	kernel size	$3 \times 3, 5 \times 5, 7 \times 7, 3 \times 3$
	Median	aperture linear size	3, 5
	Bilateral Filter	diameter, sigmaColor, sigmaSpace	9, 75, 75

2.5 Results

To evaluate the performance of our methodology, we try to answer four research questions.

1): Is neuron coverage a good metric for test generation?

To answer this question, we try to see whether different input-output pair results in different neuron coverage. For each input image we measure the neuron coverage (see Equation 1 in Section 2.3.1) of the underlying models and the corresponding output. As discussed in Section 2.4, corresponding to an input image, each model outputs a steering direction (left (+ve) / right (-ve)) and a steering angle as shown in Table 1 (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 [82] between neuron coverage and steering angle. This is a non-parametric measure to compute monotonic association between the two variables [40]. Correlation with positive statistical significance suggests that the steering angle increases with increasing neuron coverage and vice versa. Table 3 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.

Table 3: 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	-0.10 (***)	left (+ve) > right (-ve) (***)	negligible
	CNN	0.28 (***)	left (+ve) < right (-ve) (***)	negligible
	LSTM	-0.10 (***)	left (+ve) > right (-ve) (***)	negligible
Rambo	Overall	-0.11 (***)	left (+ve) < right (-ve) (***)	negligible
	S1	-0.19 (***)	left (+ve) < right (-ve) (***)	large
	S2	0.10 (***)	not significant	negligible
	S3	-0.11 (***)	not significant	negligible
Epoch	N/A	0.78 (***)	left (+ve) < right (-ve) (***)	small

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

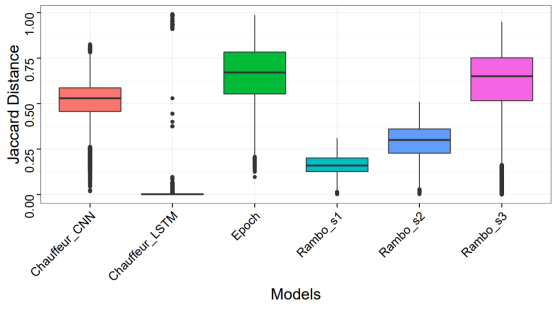
Steering direction. To measure the association between neuron coverage and steering direction, we check whether the coverage varies between right and left steering direction. We use the Wilcoxon nonparametric test as the steering direction can only have two values (left and right). Our results confirm 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.

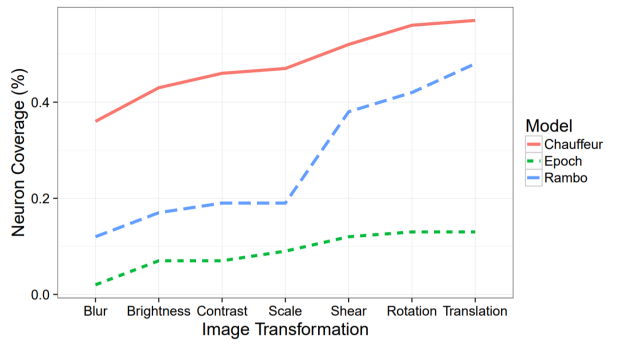
2): Do different realistic image transformations activate different neurons?

We investigate whether synthetic images generated by applying different realistic image transformations on existing input images can activate different neurons. 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_1 respectively. We measure the dissimilarities between N_1 and N_2 by measuring their Jaccard distance: $1 - \frac{|N_1 \cap N_2|}{|N_1 \cup N_2|}$.



(a) 2.1 Difference in neuron coverage caused by different image transformations



(b) 2.2 Average cumulative neuron coverage per input image

Figure 2: 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.

Figure 2.1 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.

3): Will combining different image transformations increase neuron coverage further?

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, 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 4 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 $(13,080 - 13,008)/13,008 = 0.55\%$.

Table 4: 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	% increase of guided *w.r.t.* 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%

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 , Epoch , and Rambo models respectively and measure their collective neuron coverage. As shown in Table 4, 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.

4): Can we automatically detect erroneous behaviors using metamorphic relations?

To detect erroneous behaviors we can identify the transformed images whose outputs violate the metamorphic relation defined in Equation 2. 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.

$$\mid MSE_{(trans,param)} - MSE_{org} \mid \ \leq \epsilon \tag{3}$$

Thus, we only choose the transformations that obey Equation 3 for counting erroneous behaviors. Table 5 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.

Table 5: Number of erroneous behaviors reported by DeepTest across all tested models at different thresholds

λ (see Eqn. 2)	Simple Tranformation ϵ (see Eqn. 3) 0.01	0.02	0.03	0.04	0.05	Composite Transformation Fog	Rain	Guided Search
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 6: 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

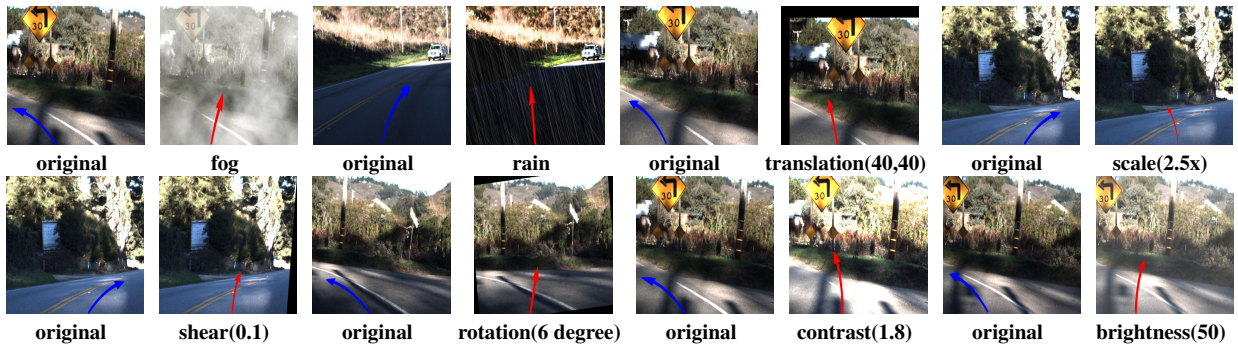


Figure 3: Sample images showing erroneous behaviors detected by DeepTest using synthetic images. For **original** images the arrows are marked in blue, while for the synthetic images they are marked in **red**. More such samples can be viewed at `https://deeplearningtest.github.io/deepTest/`.

Table 6 further elaborates the result for different models for $\lambda = 5$ and $\epsilon = 0.03$, as highlighted in Table 5. 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 3 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 6 and report the false positives in Figure 4. It also shows two synthetic images (the corresponding original images) where DeepTest incorrectly reports erroneous behaviors while the model's output is indeed safe.

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

original **translation(50,50), epoch** **original** **shear(0.4), rambo**

Figure 4: Sample false positives produced by DeepTest for $\lambda = 5$, $\epsilon = 0.03$

Table 7: 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

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 [11] 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 7.

2.6 Conclusion

We proposed and evaluated DeepTest, a tool for automated testing of DNN-based self-driving 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 software by customizing the transformations and metamorphic relations.

2.7 Future work

Future work may include repairing these erroneous behaviours by retraining self-driving car DNNs with these new generated inputs, introducing more sensors inputs besides camera input and exploring better coverage metric than neuron coverage.

3 DeepInspect

3.1 Introduction

In this work, we propose DeepInspect to test DNN-based image classifiers on class property violations. We found that many of the reported erroneous cases in popular DNN image classifiers occur because the trained models confuse one class with another or show biases towards some classes over others. These bugs usually violate some class properties of one or more of those classes. Most

DNN testing techniques focus on per-image violations, so fail to detect class-level confusions or biases.

We developed a testing technique to automatically detect class-based confusion and bias errors in DNN-driven image classification software. We evaluated our implementation, DeepInspect, on several popular image classifiers with precision up to 100% (avg. 72.6%) for confusion errors, and up to 84.3% (avg. 66.8%) for bias errors. DeepInspect found hundreds of classification mistakes in widely-used models, many exposing errors indicating confusion or bias.

3.2 Background

Image classifications include single-label classification and multi-label classification. DeepInspect is designed and proposed to work on both settings.

(i) **Single-label Classification.** In a traditional single-label classification problem, each datum is associated with a single label l from a set of disjoint labels L where $|L| > 1$. If $|L| = 2$, the classification problem is called a binary classification problem; if $|L| > 2$, it is a multi-class classification problem [90]. Among some popular image classification datasets, MNIST, CIFAR-10/CIFAR-100 [47] and ImageNet [78] are all single-label, where each image can be categorized into only one class or outside that class.

(ii) **Multi-label Classification.** In a multi-label classification problem, each datum is associated with a set of labels Y where $Y \subseteq L$. COCO[51] and imSitu[94] are popular datasets for multi-label classification. For example, an image from the COCO dataset can be labeled as *car, person, traffic light and bicycle*. A multi-label classification model is supposed to predict all of *car, person, traffic light and bicycle* from a single image that shows all of these kinds of objects.

3.3 Methodology

Figure 5 shows the DeepInspect workflow. DeepInspect works in a real-world setting where a customer gets a pre-trained model and tests its performance in a sample production scenario before deployment. The customer has white-box access to the model to profile, although all the data in the production system can be *unlabeled*. In the absence of ground truth labels, the classes are defined by the *predicted labels*. These predicted labels are used as class references as DeepInspect tries to detect confusion and bias errors among the classes.

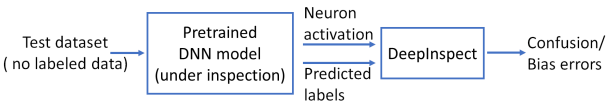


Figure 5: **DeepInspect Workflow**

DeepInspect tracks the activated neurons per class and reports a potential class-level violation if the class-level activation-patterns are too similar between two classes. Such reported errors will help customers evaluate how much they can trust the model's results related to the affected classes. Once these errors are reported back to the developers, they can focus their debugging and fixing effort on these classes.

3.3.1 Neuron Activation Probability

Before we describe DeepInspect's methodology in detail, we introduce definitions that we use in the rest of the paper. The following table shows our notation.

All neurons set	$N = \{N_1, N_2, ...\}$
Activation function	$out(N, c)$ returns output for neuron N , input c.
Activation threshold	Th

Neuron Activation Probability: Leveraging how *frequently* a neuron N_j is activated by all the members from a class C_i, this metric estimates the probability of a neuron N_j to be activated by C_i. Thus, we define: $P(N_j \mid C_i) = \dfrac{|\{c_{ik} \mid \forall c_{ik} \in C_i, out(N_j, c_{ik}) > Th\}|}{|C_i|}$

We then construct a $n \times m$ dimensional *neuron activation probability matrix*, ρ, (n is the number of neurons and m is the number of classes) with its ij-th entry being $P(N_j \mid C_i)$.

$$\rho = \begin{array}{c} \\ N_1 \\ \dots \\ N_j \\ \dots \\ N_n \end{array} \begin{array}{c} \begin{array}{cccccc} C_1 & \dots & C_i & \dots & C_m \end{array} \\ \begin{pmatrix} p_{11} & & & & p_{1m} \\ \dots & & & & \\ p_{j1} & & \dots & & p_{jm} \\ \dots & & & & \\ p_{n1} & & & & p_{nm} \end{pmatrix} \end{array} \tag{4}$$

This matrix captures how a model interacts with a set of input data. The column vectors ($\rho_{\alpha m}$) represent the interaction of a class C_m with the model. Note that, in our setting, Cs are predicted labels.

Since Neuron Activation Probability Matrix (ρ) is designed to represent each class, it should be able to distinguish between different Cs. Next, we use this metric to detect confusion errors and bias errors.

3.3.2 Finding Confusion Errors

First, we compute the confusion score between two classes as the euclidean distance between their two probability vectors:

$$\text{NAPVD}(a,b) = \Delta(a,b) = ||\rho_{\alpha a} - \rho_{\alpha b}||_2 = \sqrt{\sum_{i=1}^{n} \left(P(N_i|a) - P(N_i|b)\right)^2} \tag{5}$$

If the Δ value is less than some pre-defined threshold (conf_th) for two pairs of classes, the model will potentially make mistakes in distinguishing one from another, which results in confusion errors. This Δ is called NAPVD (Neuron Activation Probabiliy Vector Distance).

3.3.3 Finding Bias Errors

We propose an inter-class distance based metric to calculate the bias learned by the model. We define the bias between two classes a and b over a third class c as follows:

$$bias(a, b, c) := \frac{|\Delta(c, a) - \Delta(c, b)|}{\Delta(c, a) + \Delta(c, b)} \tag{6}$$

If a model treats objects of classes a and b similarly under the presence of a third object class c, a and b should have similar distance *w.r.t.* c in the embedded space ρ; thus, the numerator of the above equation will be small. Intuitively, the model's output can be more influenced by the nearer object classes, *i.e.* if a and b are closer to c. Thus, we normalize the disparity between the two distances to increase the influence of closer classes.

This bias score is used to measure how differently the given model treats two classes in the presence of a third object class. An **average bias** (abbreviated as avg_bias) between two objects a and b for all class objects O is defined as:

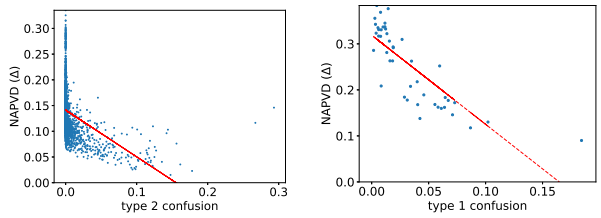
$$avg\_bias(a, b) := \frac{1}{|O| - 2} \sum_{c \in O, c \neq a, b} bias(a, b, c) \tag{7}$$

The above score captures the overall bias of the model between two classes. If the bias score is larger than some pre-defined threshold, we report potential *bias errors*.

3.4 Results

3.4.1 Detect confusion errors

In this section, We evaluate how well DeepInspect can detect confusion errors. *i.e.*, Type1/Type2 confusions.



(a) COCO dataset + ResNet-50 (b) Robust CIFAR-10 Small

Figure 6: **Strong negative Spearman correlation (-0.55 and -0.86) between NAPVD and ground truth confusion scores.**

Strong correlation has been found between NAPVD and ground truth confusion score for all 8 experimental settings. Figure 13 gives examples on COCO and CIFAR-10. These results indicate that NAPVD can be used to detect confusion errors—lower NAPVD means more confusion.

By default, DeepInspect reports all the class-pairs with NAPVD scores one standard deviation less than the mean NAPVD score as error-prone. As shown in Table 8, DeepInspect reports errors at high recall under most settings. Specifically, on CIFAR-100 and robust CIFAR-10 ResNet, DeepInspect can report errors as high as 71.8%, and 100%, respectively. DeepInspect has identified thousands of confusion errors.

Table 8: **DeepInspect performance on detecting confusion errors**

		NAPVD < mean-1std				Top 1%			
		TP	**FP**	**Precision**	**Recall**	**TP**	**FP**	**Precision**	**Recall**
COCO	DeepInspect	138	256	0.350	0.775	31	0	1	0.174
	MODE	126	382	0.248	0.708	26	5	0.839	0.146
	random	22	372	0.056	0.124	1	30	0.032	0.006
COCO gender	DeepInspect	139	286	0.327	0.827	32	0	1	0.190
	MODE	125	379	0.248	0.744	30	2	0.938	0.179
	random	22	403	0.052	0.131	1	31	0.031	0.006
CIFAR-100	DeepInspect	206	584	0.261	0.718	39	10	0.796	0.136
	MODE	111	605	0.155	0.387	22	27	0.449	0.077
	random	45	745	0.057	0.157	2	47	0.041	0.007
R CIFAR-10 S	DeepInspect	4	6	0.400	0.800	-	-	-	-
	MODE	3	4	0.429	0.600	-	-	-	-
	random	1	9	0.100	0.200	-	-	-	-
R CIFAR-10 L	DeepInspect	3	4	0.430	0.600	-	-	-	-
	MODE	3	5	0.375	0.600	-	-	-	-
	random	0	7	0	0	-	-	-	-
R CIFAR-10 R	DeepInspect	5	3	0.625	1	-	-	-	-
	MODE	1	3	0.250	0.200	-	-	-	-
	random	0	8	0	0	-	-	-	-
ImageNet	DeepInspect	4014	69957	0.054	0.617	1073	3922	0.215	0.165
	MODE	3428	66987	0.049	0.527	1591	3404	0.319	0.245
	random	962	73009	0.013	0.148	65	4930	0.013	0.010
imSitu	DeepInspect	48	58	0.453	0.165	31	19	0.620	0.107
	random	6	100	0.057	0.020	2	48	0.040	0.007

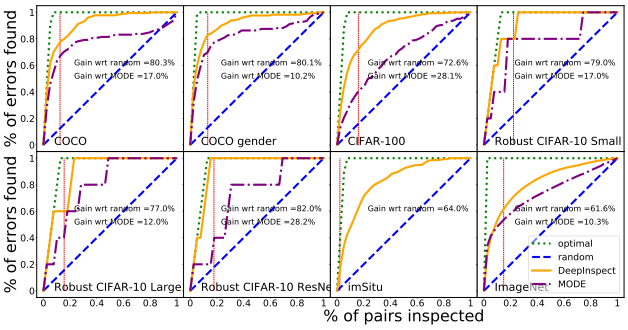
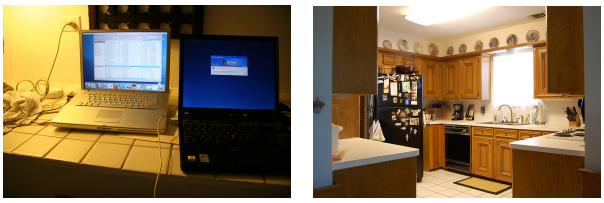


Figure 7: **AUCEC plot of Type1/Type2 Confusion errors in three different settings. The red vertical line marks 1-standard deviation less from mean NAPVD score. DeepInspect marks all class-pairs with NAPVD scores less than the red mark as potential errors.**



(a) (keyboard,mouse) (b) (oven,microwave)

Figure 8: **Confusion errors identified in COCO model. In each pair the second object is mistakenly identified by the model.**

If higher precision is wanted, a user can choose to inspect only a small set of confused pairs based on NAPVD. As also shown in Table 8, when only the top1% confusion errors are reported, a much higher precision is achieved for all the datasets. In particular, DeepInspect identifies 31 and 39 confusion errors for the COCO model and the CIFAR-100 model with 100% and 79.6% precision, respectively. The trade-off between precision and recall can be found on the cost-effective curves shown on Figure 7, which show overall performance of DeepInspect at different inspection cutoffs. Overall, *w.r.t.* a random baseline mode, DeepInspect is gaining AUCEC performance from 61.6% to 85.7%; *w.r.t.* a MODE baseline mode, DeepInspect is gaining AUCEC performance from 10.2% to 28.2%.

Figure 8 and Figure 9 show some real confusion errors identified by DeepInspect in the COCO and the ImageNet settings. In particular, as shown in Figure 8a, when there is only a keyboard but no mouse in the image, the COCO model reports both. Similarly, Figure 9a shows confusion errors on (cello, violin). There are several cellos in this image, but the model predicts violin.

DeepInspect can successfully find confusion errors with precision 21% to 100% at top1% for both single- and multi-object classification tasks. DeepInspect also finds confusion errors in robust models.

(a) (cello, violin) (b) (library, bookshop)

Figure 9: **Confusion errors identified in the ImageNet model. For each pair, the second object is mistakenly identified by the model.**

3.4.2 Detect bias errors

In this section, we evaluate how well DeepInspect can detect bias errors.

Table 9: **DeepInspect performance on detecting bias errors**

		avg_bias > mean+1std				Top 1%			
		TP	**FP**	**Precision**	**Recall**	**TP**	**FP**	**Precision**	**Recall**
COCO	DeepInspect	249	278	0.472	0.759	24	8	0.75	0.073
	MODE	145	324	0.309	0.442	12	20	0.375	0.037
	random	54	472	0.103	0.167	3	28	0.103	0.010
COCO gender	DeepInspect	218	325	0.401	0.568	17	16	0.515	0.044
	MODE	151	328	0.315	0.393	13	20	0.394	0.034
	random	64	478	0.118	0.168	3	28	0.118	0.010
CIFAR-100	DeepInspect	310	543	0.363	0.380	29	21	0.580	0.036
	MODE	69	315	0.180	0.085	5	45	0.100	0.001
	random	140	711	0.165	0.172	8	41	0.165	0.010
R CIFAR-10 S	DeepInspect	7	4	0.636	0.778	-	-	-	-
	MODE	3	10	0.231	0.333	-	-	-	-
	random	2	8	0.200	0.222	-	-	-	-
R CIFAR-10 L	DeepInspect	6	7	0.462	0.667	-	-	-	-
	MODE	8	14	0.364	0.889	-	-	-	-
	random	2	9	0.200	0.267	-	-	-	-
R CIFAR-10 R	DeepInspect	6	3	0.667	0.667	-	-	-	-
	MODE	8	14	0.364	0.889	-	-	-	-
	random	1	7	0.200	0.200	-	-	-	-
ImageNet	DeepInspect	26704	48913	0.353	0.330	3253	1742	0.651	0.040
	MODE	23881	47503	0.335	0.295	2355	2640	0.471	0.029
	random	12234	63381	0.162	0.151	808	4186	0.162	0.010
imSitu	DeepInspect	408	311	0.567	0.718	43	8	0.843	0.076
	random	80	638	0.112	0.142	5	44	0.112	0.010

Table 9 shows the precision and recall in detecting bias errors. At cutoff Top1%(avg_bias), DeepInspect detects suspicious pairs with precision as high as 75% and 84% for COCO and imSitu, respectively. At cutoff mean(avg_bias)+standard deviation(avg_bias), DeepInspect has high recall but lower precision: the recall is 75.9% and 71.8% for COCO and imSitu. DeepInspect reports 657(=249+408) total true bias bugs across the two models and outperforms the random baseline by a large margin at both cutoffs.

As shown in Figure 10, DeepInspect outperforms the baseline by a large margin. The AUCEC gains of DeepInspect are from 37.1% to 76.1% w.r.t. the random baseline and from 6.0% to 41.9% w.r.t. the MODE baseline across the 8 settings. DeepInspect's performance is close to the optimal curve under some settings, specifically the AUCEC gains of the optimal over DeepInspect are only 7.11% and 7.95% under the COCO and ImSitu settings, respectively.

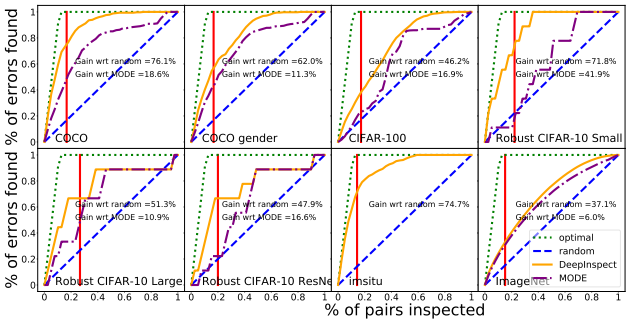


Figure 10: **Bias errors detected *w.r.t.* the ground truth of avg_cd beyond one standard deviation from mean.**

Inspired by [102], which shows bias exists between men and women in COCO for the gender image captioning task, we analyze the most biased third class c for a and b being men and women. As shown in Figure 11, we found that sports like skiing, snowboarding, and surfboarding are more closely associated with men and thus misleads the model to predict the women in the images as men. Figure 12 shows the bias errors identified on imSitu dataset, where we found that the model tends to associate the class "inside" with women while associating the class "outside" with men. Overall, DeepInspect can successfully find bias errors for both single- and multi-label classification tasks, and even for the robust models, from 52% to 84% precision at top1%.

3.5 Conclusion

Our testing tool for DNN-based image classifiers, DeepInspect, automatically detects confusion and bias errors in classification models. We applied DeepInspect to six different popular image

Figure 11: **The model classifies the women in these pictures as men in the COCO dataset.**

Figure 12: **The model classifies the man in the first figure to be a woman and the woman in the second figure to be a man.**

classification datasets and eight pretrained DNN models, including three so-called relatively more robust models. We show that DeepInspect can successfully detect class-level violations for both single- and multi-label classification models with high precision.

3.6 Future work

Repairing confusion and bias errors for DNN-based image classifiers will be in future work.

4 Repair Confusion and Bias Errors

4.1 Introduction

Our previous work DeepInspect shows that DNN based image classifiers are susceptible to confusion and bias errors. Even for robust trained model, DeepInspect still identifies lots of confusion errors and bias errors. This work is a direct future work of DeepInspect. The goal is to reduce confusions and bias for target pairs and triples.

NAPVD (Neuron Activation Probability Vector Distance) is proposed in DeepInspect to measure the class level relation perceived by a pre-trained DNN model. It defined the probability vector of each class from a given model under test as follows,

$$P(C) = [A(n_1)/N, A(n_2)/N, ..., A(n_t)/N],$$

where N is number of images in class C, $A(n_i)$ means how many times the neuron n_i is activated when feeding these N images to the model under test. The distance between two classes is the L2 norm between these two classes' probability vectors. DeepInspect shows that the smaller $\text{NAPVD}(x, y)$, the model will be more confused between x and y. The larger $\text{NAPVD}(x, z) - \text{NAPVD}(y, z)$, the model will be bias towards y than x in presence of z. It leveraged this correlation to detect confusion and bias errors.

However NAPVD distance metric cannot be used in loss function because it is not differentiable. We proposed a differentiable distance metric based on NAPVD and propose a repairing approach by increasing the distance between two classes during retraining the model to reduce the confusion errors. We evaluate our approaches on both single-label and multi-label classification models and datasets. Our results show that our approach effectively reduce confusion errors with very slight accuracy reduce.

4.2 Methodology

4.2.1 Differentiable Distance Metric

We proposed the following definition of perception of class C from a model under test.

$$P_{new}(C) = [S(n_1)/N, S(n_2)/N, ..., S(n_t)/N],$$

where $S(n_i)$ is the sum of each output of neuron n_i, given N input images. Then, the distance metric we proposed is as follows,

$$D_{new}(x, y) = L2\_norm(P_{new}(x), P_{new}(y))$$

In this definition, both $P_{new}(C)$ and D_{new} are differentiable.

4.2.2 Repairing Confusion and Bias Errors

We proposed the following two loss functions to reduce confusion errors and bias errors respectively,

$$Loss_{confusion} = Loss_{orig} - \lambda D_{new}(x, y)$$

$$Loss_{bias} = Loss_{orig} + \lambda \, abs(D_{new}(x, z) - D_{new}(y, z))$$

To optimize on $Loss_{confusion}$ is to reduce the original loss and at the same time increase the distance between class x and class y. Similarly, to optimize on $Loss_{bias}$ means to reduce the original loss and at the same time reducing the distance difference between (x, z) and (y, z).

4.3 Initial Results

We evaluate our approaches on a state-of-the-art ResNet-50 model trained using CutMix[96] in CIFAR-10/CIFAR-100[47] dataset and a pre-trained ResNet-50 model[102] in COCO[51] dataset. In all evaluation, we set λ to be 0.5. Exploring how λ affects the performance will be in future work.

CIFAR-10/CIFAR-100

Original model is trained for 300 epochs. Both regular retrain and repair retrain take another 60 epochs. However, repair retrain leveraged our new proposed loss function. As shown in Table 10 and Table 11, the repair retrain can effectively reduce the confusion or bias with very slight accuracy reduce. For CIFAR-100 model, the bias repair even increases the overall accuracy.

Table 10: **Repair CIFAR-10 model**

(a) Reduce (Dog, Cat) confusion

	original model	regular retrain	repair retrain
Accuracy	92.93%	92.78%	92.73%
Dog→Cat	0.066	0.078	0.059
Cat→Dog	0.069	0.069	0.062
Avg	0.068	0.074	0.061

(b) Reduce (Dog, Cat, Car) bias

	original model	regular retrain	repair retrain
Accuracy	92.93%	92.9%	92.88%
Dog-Cat	0.068	0.065	0.057
Dog-Car	0.001	0.001	0.00
Bias	0.067	0.064	0.056

Table 11: **Repair CIFAR-100 model**

(a) Reduce (Otter, Seal) confusion

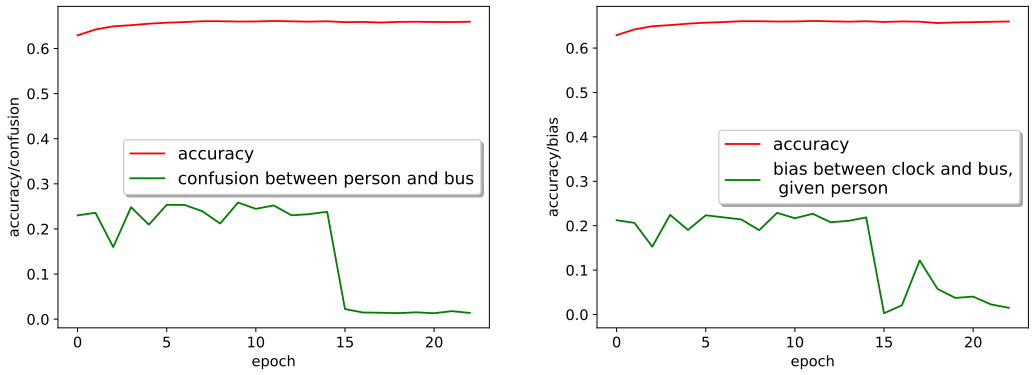
	original model	regular retrain	repair retrain
Accuracy	70.87%	71.26%	70.73%
Otter→Seal	0.16	0.10	0.11
Seal→Otter	0.14	0.21	0.11
Avg	0.15	0.15	0.11

(b) Reduce (Otter, Seal, Apple) bias

	original model	regular retrain	repair retrain
Accuracy	70.87%	71.97%	71.94%
Dog-Cat	0.15	0.12	0.08
Dog-Car	0.00	0.00	0.00
Bias	0.15	0.12	0.08

MS-COCO

We introduced our proposed new loss function to repair the model starting from epoch 15, Figure 13 shows that after epoch 15, the confusion and bias are reduced significantly.



(a) Reduce (person, bus) confusion (b) Reduce (person, clock, bus) bias

Figure 13: **Repair MS-COCO model**

4.4 Conclusion and future work

The preliminary results show that our approach can effectively reduce confusion errors and bias errors. In future work, we will conduct a comprehensive evaluation of confusion and bias reduction for different pairs/triplets on more datasets. We will also analyze how λ affects the performance.

5 Research plan

5.1 Plan for completion of the research

Table 12 shows my plan for completion of the research.

Timeline	Work	Progress
May. 2018	DeepTest ICSE'18 publication	completed
June. 2020	DeepInspect ICSE'20 publication	completed
Nov. 2020	Repair confusion errors FSE'20 publication	completed
March. 2021	Repair confusion and bias errors	ongoing
April. 2021	Thesis writting	
May. 2021	Thesis defense	

Table 12: Plan for completion of my research

Thus, I plan to defend my thesis in May 2021.

References

[1] 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] 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] Open source computer vision library. https://github.com/itseez/opencv, 2015.

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

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

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

[7] 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.

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

[9] Tesla autopilot. https://www.tesla.com/autopilot, 2016.

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

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

[12] Baidu apollo. https://github.com/ApolloAuto/apollo, 2017.

[13] 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.

[14] The numbers don't lie: Self-driving cars are getting good. https://www.wired.com/2017/02/california-dmv-autonomous-car-disengagement, 2017.

[15] Software 2.0. https://medium.com/@karpathy/software-2-0-a64152b37c35, 2017.

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

[17] 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. An orchestrated survey of methodologies for automated software test case generation. *Journal of Systems and Software*, 86(8):1978–2001, 2013.

[18] Solon Barocas, Moritz Hardt, and Arvind Narayanan. *Fairness and Machine Learning*. fairmlbook.org, 2018. http://www.fairmlbook.org.

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

[20] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Computer Vision and Pattern Recognition*, 2017.

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

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

[23] Yuriy Brun and Alexandra Meliou. Software fairness. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, pages 754–759, New York, NY, USA, 2018. ACM.

[24] T. Calders, F. Kamiran, and M. Pechenizkiy. Building classifiers with independency constraints. In *2009 IEEE International Conference on Data Mining Workshops*, pages 13–18, Dec 2009.

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

[26] Tsong Y Chen, Shing C Cheung, and Shiu Ming Yiu. 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, 1998.

[27] François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[28] California DMV. Autonomous vehicle disengagement reports. `https://www.dmv.ca.gov/portal/dmv/detail/vr/autonomous/disengagement_report_2016`, 2016.

[29] Yinpeng Dong, Hang Su, Jun Zhu, and Fan Bao. Towards interpretable deep neural networks by leveraging adversarial examples. *arXiv preprint arXiv:1708.05493*, 2017.

[30] Michele Donini, Luca Oneto, Shai Ben-David, John Shawe-Taylor, and Massimiliano Pontil. Empirical risk minimization under fairness constraints. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pages 2796–2806, 2018.

[31] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard S. Zemel. Fairness through awareness. *In Proceedings of the Innovations in Theoretical Computer Science Conference*, abs/1104.3913:214–226, 2012.

[32] Ivan Evtimov, Kevin Eykholt, Earlence Fernandes, Tadayoshi Kohno, Bo Li, Atul Prakash, Amir Rahmati, and Dawn Song. Robust physical-world attacks on machine learning models. *arXiv preprint arXiv:1707.08945*, 2017.

[33] Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410*, 2017.

[34] Sainyam Galhotra, Yuriy Brun, and Alexandra Meliou. Fairness testing: testing software for discrimination. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 498–510. ACM, 2017.

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

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

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

[38] Jianmin Guo, Yu Jiang, Yue Zhao, Quan Chen, and Jiaguang Sun. Dlfuzz: Differential fuzzing testing of deep learning systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 739–743, 2018.

[39] Moritz Hardt, Eric Price, and Nathan Srebro. Equality of opportunity in supervised learning. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 3323–3331, USA, 2016.

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

[41] Warren He, James Wei, Xinyun Chen, Nicholas Carlini, and Dawn Song. Adversarial example defenses: Ensembles of weak defenses are not strong. In *Proceedings of the 11th USENIX Conference on Offensive Technologies*, WOOT'17, pages 15–15, Berkeley, CA, USA, 2017. USENIX Association.

[42] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*, 2017.

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

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

[45] Michael P. Kim, Omer Reingold, and Guy N. Rothblum. Fairness through computationally-bounded awareness. *32nd Conference on Neural Information Processing Systems (NeurIPS 2018)*, 2018.

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

[47] Alex Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012.

[48] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *Workshop track at International Conference on Learning Representations (ICLR)*, 2017.

[49] Matt J Kusner, Joshua Loftus, Chris Russell, and Ricardo Silva. Counterfactual fairness. In *Advances in Neural Information Processing Systems 30*, pages 4066–4076, 2017.

[50] Alexandre Louis Lamy, Ziyuan Zhong, Aditya Krishna Menon, and Nakul Verma. Noise-tolerant fair classification. *CoRR*, abs/1901.10837, 2019.

[51] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[52] Zachary C Lipton. The mythos of model interpretability. *Proceedings of the 33rd International Conference on Machine Learning Workshop*, 2016.

[53] Jiajun Lu, Hussein Sibai, Evan Fabry, and David Forsyth. No need to worry about adversarial examples in object detection in autonomous vehicles. In *Spotlight Oral Workshop at Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017.

[54] Binh Thanh Luong, Salvatore Ruggieri, and Franco Turini. k-nn as an implementation of situation testing for discrimination discovery and prevention. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 502–510. ACM, 2011.

[55] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang. Deepgauge: Multi-granularity testing criteria for deep learning systems. pages 120–131, 2018.

[56] Chengzhi Mao, Ziyuan Zhong, Junfeng Yang, Carl Vondrick, and Baishakhi Ray. Metric learning for adversarial robustness. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 478–489. Curran Associates, Inc., 2019.

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

[58] Aditya Krishna Menon and Robert C. Williamson. The cost of fairness in binary classification. In *Conference on Fairness, Accountability and Transparency, FAT 2018, 23-24 February 2018, New York, NY, USA*, pages 107–118, 2018.

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

[60] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[61] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 2017.

[62] Christian Murphy, Gail E Kaiser, Lifeng Hu, and Leon Wu. Properties of machine learning applications for use in metamorphic testing. In *SEKE*, volume 8, pages 867–872, 2008.

[63] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

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

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

[66] Augustus Odena, Catherine Olsson, David Andersen, and Ian Goodfellow. TensorFuzz: Debugging neural networks with coverage-guided fuzzing. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4901–4911, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[67] Nicolas Papernot, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Fartash Faghri, Alexander Matyasko, Karen Hambardzumyan, Yi-Lin Juang, Alexey Kurakin, Ryan Sheatsley, et al. cleverhans v2. 0.0: an adversarial machine learning library. *arXiv preprint arXiv:1610.00768*, 2016.

[68] Nicolas Papernot and Patrick McDaniel. Extending defensive distillation. *arXiv preprint arXiv:1705.05264*, 2017.

[69] Nicolas Papernot and Patrick McDaniel. Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning. *arXiv preprint arXiv:1803.04765*, 2018.

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

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

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

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

[74] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. pages 1–18, 2017.

[75] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Towards practical verification of machine learning: The case of computer vision systems. *arXiv preprint arXiv:1712.01785*, 2017.

[76] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. *6th International Conference on Learning Representations (ICLR)*, 2018.

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

[78] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[79] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, and Michael Young. Machine learning: The high interest credit card of technical debt. 2014.

[80] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 618–626, Oct 2017.

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

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

[83] Youcheng Sun, Xiaowei Huang, and Daniel Kroening. Testing deep neural networks. *arXiv preprint arXiv:1803.04792*, 2018.

[84] Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. Concolic testing for deep neural networks. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE 2018, pages 109–119, New York, NY, USA, 2018. ACM.

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

[86] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[87] Yuchi Tian. Repairing confusion and bias errors for dnn-based image classifiers. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1699–1700, 2020.

[88] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering*, pages 303–314, 2018.

[89] Yuchi Tian, Ziyuan Zhong, Vicente Ordonez, Gail Kaiser, and Baishakhi Ray. Testing dnn image classifiers for confusion & bias errors. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 1122–1134, 2020.

[90] Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)*, 3(3):1–13, 2007.

[91] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. 2018.

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

[93] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*, 2017.

[94] Mark Yatskar, Luke Zettlemoyer, and Ali Farhadi. Situation recognition: Visual semantic role labeling for image understanding. In *Conference on Computer Vision and Pattern Recognition*, 2016.

[95] X. Yuan, P. He, Q. Zhu, and X. Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–20, 2019.

[96] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *International Conference on Computer Vision (ICCV)*, 2019.

[97] Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez Rodriguez, and Krishna P. Gummadi. Fairness beyond disparate treatment & disparate impact: Learning classification without disparate mistreatment. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1171–1180, 2017.

[98] Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez Rodriguez, and Krishna P Gummadi. Fairness constraints: Mechanisms for fair classification. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *(AISTATS) 2017*. JMLR, 2017.

[99] Rich Zemel, Yu Wu, Kevin Swersky, Toni Pitassi, and Cynthia Dwork. Learning fair representations. In *Proceedings of the 30th International Conference on Machine Learning*, pages 325–333, 2013.

[100] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. Deeproad: Gan-based metamorphic autonomous driving system testing. *arXiv preprint arXiv:1802.02295*, 2018.

[101] Quan-shi Zhang and Song-Chun Zhu. Visual interpretability for deep learning: a survey. *Frontiers of Information Technology & Electronic Engineering*, 19(1):27–39, 2018.

[102] Jieyu Zhao, Tianlu Wang, Mark Yatskar, Vicente Ordonez, and Kai-Wei Chang. Men also like shopping: Reducing gender bias amplification using corpus-level constraints. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2941–2951, 2017.

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

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