Testing Deep Neural Network based Image Classifiers

Yuchi Tian*
Columbia University
New York, USA
yuchi.tian@columbia.edu

Ziyuan Zhong*
Columbia University
New York, USA
ziyuan.zhong@columbia.edu

Vicente Ordonez
University of Virginia
Charlottesville, USA
vicente@virginia.edu

Baishakhi Ray
Columbia University
New York, USA
rayb@cs.columbia.edu

***Abstract*—Image classification is an important task in today's world with many applications from socio-technical to safety-critical domains. The recent advent of Deep Neural Network (DNN) is the key behind such a wide-spread success. However, such wide adoption comes with the concerns about the reliability of these systems, as several erroneous behaviors have already been reported in many sensitive and critical circumstances. Thus, it has become crucial to rigorously test the image classifiers to ensure high reliability.**

Many reported erroneous cases in popular neural image classifiers appear because the models often confuse one class with another, or show biases towards some classes over others. These errors usually violate some group properties. Most existing DNN testing and verification techniques focus on per image violations and thus fail to detect such group-level confusions or biases. In this paper, we design, implement and evaluate DeepInspect, a white box testing tool, for automatically detecting confusion and bias of DNN-driven image classification applications. We evaluate DeepInspect using popular DNN-based image classifiers and detect hundreds of classification mistakes. Some of these cases are able to expose potential biases of the network towards certain populations. DeepInspect further reports many classification errors in state-of-the-art robust models.

***Index Terms*—testing, deep learning, white box, DNN**

I. Introduction

Image categorization is a well-studied problem in computer vision, where a model is trained to classify an image into single or multiple predefined categories [1]. It has a plethora of applications in safety-critical domains like self-driving cars, health care, *etc.* Even in our day-to-day life we often use image classifiers: for example, Google Photo search, Facebook image tagging, *etc.* With the advent of Deep Neural Networks (DNN), such image classification tasks have seen major breakthroughs over the past few years—sometimes even matching human-level accuracy under some conditions [2].

In spite of such spectacular success, we often encounter high-impact classification errors made by these models, as shown in Table I. For example, in 2015, Google faced huge backlash due to a notorious error in its photo-tagging app, which tagged pictures of dark-skinned people as "gorillas" [3]. After manual investigation of some related public reports, we find two main causes behind such mistakes: (i) **Confusion**: The model cannot differentiate one class from another. For example, it has been recently reported that Google Photos confuse skier and mountain [4], and (ii) **Bias**: The model shows disparate outcomes between two related groups. For example, Zhao *et al.* [5] in their paper "Men also like shopping" find classification bias towards women on activities like shopping, cooking, washing, *etc.*

These errors are specific to a class of images rather than any particular input image—at a high level, the intuition is that some class properties are getting violated in such cases. For example, in the case of bias reported by Zhao *et al.* [5], a DNN model should not have different error rates while classifying the gender of a person in the shopping category. Thus, unlike individual image properties, this is a class property defined over all the shopping images with men and women. Any violation of such property affects the whole class, *e.g.,* man is more likely to be predicted as woman when he is shopping, although many individual images in this category can still be predicted correctly.

Due to the lack of such formally specified properties, while designing a DNN, developers usually follow some mental models of informal specifications; an *error* occurs when the application "produced harmful and unexpected results" *w.r.t.* that informal specification, as observed by Google Brain researchers [9]. Without such specification, traditional software testing techniques that test *w.r.t.* some oracle will remain inadequate [10]–[12].

One simple workaround to detect class-level violations could be to simply analyze the class separations because after all the DNN models are supposed to learn this separation. To identify possible sources of confusion/bias-related errors we should check whether the class-separation is enough to distinguish clearly between two classes. In fact, traditional ML testing/validation step indirectly evaluates such class-separation by measuring the model's accuracy. However, doing this in a black-box manner is inadequate, especially for a pre-trained model, because (i) the training and testing distributions can widely differ; hence the labeled data used to validate the model might not measure the class separation accurately in a real-world test environment and miss the corner-cases, and (ii) the input space (*e.g.,* pixel space) is not well-specified to identify the class properties or their separation.

In this work, we propose a novel white box technique to capture the separation between two classes. For a set of test input images, we compute the probability of activation of a neuron per predicted class. Thus, for each class, we create a vector of neuron activations where each vector element

*The first two authors contributed equally to this work

TABLE I: **Examples of real-world errors reported in neural image classifiers**

Error Type	Name	Report Date	Outcome
Confusion	Gorilla Tag[3]	Jul 1, 2015	Black people were tagged as gorilla by Google photo app.
	Elephant is detected in a room[6]	Aug 9, 2018	Image Transplantation (*i.e.* replacing sub-region of an image by another image containing trained object) leads to mis-classification.
	Google Photo[4]	Dec 10, 2018	Google Photo confuses skier and mountain.
Bias	Nikon Camera[7]	Jan 22, 2010	Camera shows bias toward Caucasian faces when detecting people's blinks.
	Men Like Shopping[5]	July 29, 2017	Multilabel object classification models show bias towards women on activities like shopping, cooking, washing *etc.*
	Gender Shades[8]	2018	Open source face recognition services provided by IBM, Microsoft, and Face++ have higher error rates on darker-skin females for gender classification.

corresponds to a neuron activation probability. If the distance between two vectors is too close (compared to other class-vector pairs) that means the DNN under test cannot effectively distinguish the two classes because often a similar set of neurons are activated by the corresponding class members.

To this end, we propose a novel white-box test strategy, DeepInspect. We evaluate DeepInspect for both single- and multi-label classification models in 10 different settings. Our experiments demonstrate that DeepInspect, unlike existing white-box techniques, can efficiently detect both Bias and Confusion errors in popular neural image classifiers. For all the models we have tested, DeepInspect reports a large number of classification errors with high precision. We further check whether DeepInspect can detect such classification errors in state-of-the-art models designed to be robust against norm-bounded adversarial attacks; DeepInspect finds hundreds of errors proving the need for orthogonal testing strategies to detect such class-level mispredictions. Further, unlike common DNN testing techniques [13], [14], we do not need to generate additional transformed images to find these errors.

We summarize our contributions as follows:

- We design a novel white-box testing framework to automatically detect confusion and bias errors in DNN based visual recognition models for image classification.
- We implement the proposed techniques in DeepInspect and exhaustively evaluate DeepInspect and detect many errors in state-of-the art DNN models.
- We have made the errors reported by DeepInspect public at https://deeplearninginspect.github.io/DeepInspect. We plan to release the code and processed data for public use.

II. Theory

A. DNN Background

Deep Neural Networks (DNN) are popular machine learning models loosely inspired by the neural networks of human brains. A DNN model learns the logic to perform a task from a set of training examples. For example, an image recognition model learns to recognize `cows` through training with lots of sample images of cows.

A typical feed forward DNN consists of a set of connected computational units, often referred as *neurons*, which are arranged sequentially in a series of *layers*, The neurons in different layers are connected with each other through *edges*. Each edge has a corresponding weight. Each neuron applies σ, a *nonlinear activation function* (*e.g.,* ReLU [15], Sigmoid [16]), on its inputs and sends the output to the subsequent neurons. For image classification, convolutional neural networks (CNNs) [17] are typically used, which consist of layers with local spatial connectivity and sets of neurons with shared parameters across space. Since our methods are general, we will just refer to DNNs more broadly.

To build a DNN application, developers typically start with a set of pre-annotated experimental data and divide it into three sets: (i) training: to fit the model in a supervised setting (e.g., using stochastic gradient descent with gradients computed using back-propagation [18]), (ii) validation: to tune the model hyper-parameters, and (iii) evaluation: to evaluate the accuracy of the trained model *w.r.t.* to a pre-annotated test dataset. Typically, the training, validation, and testing data are drawn from the same dataset. The semantics (F) of the underlying task learned by a DNN model highly depend on the training dataset and are encoded as the weights of the edges in the network. If the training data are changed, the semantics learned by the model also change and essentially a different program is generated [19]. Thus, a final deliverable DNN application is a combination of the training data and the underlying DNN structure.

For image classification, a DNN can be trained in following two settings: (i) **Single-label Classification.** In a traditional single-label classification problem, each data 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 called a multi-class classification problem [20]. In popular image classification tasks, MNIST, CIFAR-10/CIFAR-100[21] and ImageNet[22] are all single-label classification tasks where each image can be categorized into only one class. (ii) **Multi-label Classification.** In a multi-label classification problem, each data is associated with a set of labels Y where $Y \subseteq L$. COCO[23], NUS[24] and imSitu[25] are multi-label classification tasks. For example, an image from COCO dataset can be labeled as *car, person, traffic light and bicycle*. A multi-label classification model is supposed to predict *car, person, traffic light and bicycle* from this image.

Given any single- or multi-label classification task, the DNN classifier tries to learn the decision boundary between the classes—all members of a class, say C_i, should be categorized identically irrespective of their individual features, and members of another class, say C_j, should not be categorized to C_i [26]. The DNN represents the input image in an embedded space and then use a linear classifier (*e.g.,* softmax) to classify these representations. A *class separation* estimates how well the DNN has learned to separate a class from another. If the embedded distance between two classes are too small compared to other classes or lower than some pre-defined threshold, we assume that the DNN could not separate them from each other.

B. Testing DNN Classifiers

Traditionally in ML, how well a DNN can classify and learn class-separations is tested *w.r.t.* annotated ground truth data [27]–[29]. During development time, since the training and testing samples are usually drawn from the same distribution, previous papers showed that such techniques are not adequate to test real-world corner cases [13], [14], [30]–[32]. To detect class-level violations, as shown in Table I, testing each class separately *w.r.t.* the high-level class property (*e.g.,* all the possible cow images, such as a black cow with long horns, a white cow with short horns, *etc.*, should be classified as `ranch animal`) can be a viable option. In fact, this is somewhat similar to equivalence-partition testing [33]) in traditional software engineering. However, in traditional SE it is usually enough to test one candidate per equivalent class (EC) and the boundary conditions. This is only true when the members of an EC are homogeneous. For heterogeneous class members, such as ours (*e.g.,* white cow, black cow, *etc.*), we should test each EC exhaustively [34]. But each EC can potentially have a large number of members [35] and finding representative candidates requires intelligently sampling a nonlinear, high-dimensional space, which is inherently difficult. Thus, exhaustive black-box testing is unfeasible.

Instead, we need a white-box metric, analogous to the branch/path coverage metric, that can be used to approximately represent each class. The popular approaches of white-box testing in the literature are based on DNN structure that often targets activated neurons and layers [13], [14], [30], [31]. However, these techniques are more suitable for testing the network structure (*e.g.,* how many neurons are activated) rather than class properties. Also, these techniques are more suitable for detecting image-level violations than class-level errors. For example, the method proposed in [36] can detect adversarial images at point-level but cannot be easily extended to identify violations at a group-level. In this work, we address these issues by introducing novel white-box techniques.

III. Methodology

In this section, we provide a detailed technical description of DeepInspect. Our experimental setting reflects a real-world scenario where a customer gets a pre-trained model which is running in a production system. The customer has the white box access of the model to profile, although all the data in the production system is unlabeled. Here, we primarily focus on inspecting how well a pre-trained DNN has learned the class-separation by testing it with a set of unlabeled data. In the absence of ground truth labels, the classes are defined by the predicted labels. We finally output class-pairs that are too close as compared to the rest of the class pairs and report potential confusion and bias errors accordingly. Figure 1 shows the DeepInspect workflow.

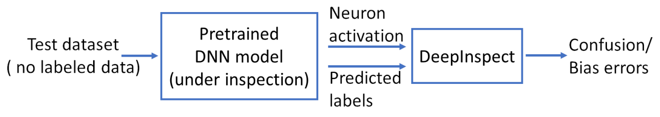


Fig. 1: **DeepInspect Workflow**

A. Definitions

Before we start describing DeepInspect methodology, we would like to introduce some definitions that we will follow in the rest of the paper.

Neural-Path (NP). For an input image x, we define *neural-path* as a *set* of neurons that are activated by x.

Neural-Path per Class (NP_C). For an class C_i, this metric represents the total number of unique neural-paths activated by all the inputs representing C_i.

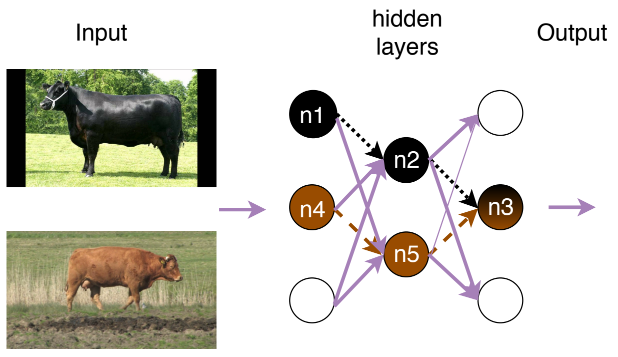


Fig. 2: **Simplified image of activated neurons by the objects of heterogeneous class cow.**

For example, consider a class `cow` containing two images: a brown cow and a black cow (see Figure 2). Let's assume that they activate two neural-paths: $\{n_1, n_2, n_3\}$ and $\{n_4, n_5, n_3\}$. Thus, the neural paths for class `cow` would be $NP_{cow} = \{\{n_1, n_2, n_3\}, \{n_4, n_5, n_3\}\}$. NP_{cow} is further represented by a vector $(n_1^1, n_2^1, n_3^2, n_4^1, n_5^1)$, where the superscripts represent the number of times each neuron is activated by C_{cow}. In fact, each C in a dataset can be expressed with such *neuron activation frequency vector*, which captures how the model interacts with that C.

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:

$$P(n_j \mid C_i) = \frac{|\{c_{ik} \mid \forall c_{ik} \in C_i, out(n_j, c_{ik}) > Th\}|}{|C_i|}$$

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 $P(n_j \mid C_i)$.

$$\rho = \begin{array}{c} \\ n_1 \\ \dots \\ n_j \\ \dots \\ n_N \end{array} \begin{array}{c} \begin{array}{ccccc} C_1 & \dots & C_i & \dots & C_M \end{array} \\ \begin{pmatrix} p_{11} & & & & p_{1M} \\ \dots & & & & \\ p_{j1} & & \dots & & p_{jM} \\ \dots & & & & \\ p_{N1} & & & & p_{NM} \end{pmatrix} \end{array} \qquad (1)$$

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 ρ is designed to represent each class, it should be able to distinguish between different Cs. Next, we use ρ to find two different classes of errors often found in DNN systems: *confusion* and *bias* (see Table I).

B. Finding Confusion Errors

In an object classification task, when the model cannot distinguish one object class from another, confusion occurs. For example, as shown in Table I, a Google photo app model confuses a skier with the mountain. Thus, finding confusion errors means checking how well the model can distinguish between objects of different classes. An error happens when the model under test classifies an object with a wrong class, or for multi-label classification task, predicts a class even though no object from that class is present in the test image.

We argue that the model makes these errors because during the training process the model has not learned to distinguish well between the two classes, say a and b. Therefore, the neurons activated by these objects are similar and the column vectors corresponding to these classes: $\rho_{\alpha a}$ and $\rho_{\alpha b}$ will be very close to each other. Thus, we compute the confusion score between two classes as the distance between their two probability vectors:

$$\begin{aligned}\text{NAPVD}(a,b) = \Delta(a,b) &= \rho_{\alpha a} - \rho_{\alpha b} \\ &= \sqrt{\sum_{n=n_1}^{n_N} \left(P(n|a) - P(n|b)\right)^2}\end{aligned} \tag{2}$$

If the Δ value is less than some pre-defined threshold 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).

C. Finding Bias

In an object classification task, bias occurs if the model under test shows disparate outcomes between two related groups. For example,

we find that ResNet-34 pretrained by imSitu dataset, often mis-classifies a man with a baby as `woman`. We observe that in the embedded matrix ρ, $\Delta(baby, woman)$ is much smaller than $\Delta(baby, man)$. Therefore, during testing, whenever the model finds an image with a baby, it is biased towards associating the baby image with a woman. Based on this observation, 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:

$$bias(a,b,c) := \frac{|\Delta(c,a) - \Delta(c,b)|}{\Delta(c,a) + \Delta(c,b)} \tag{3}$$

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:

$$avg\_bias(a,b) := \frac{1}{|O|-2} \sum_{c \in O, c \neq a,b} \text{bias}(a,b,c) \tag{4}$$

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*.

Using the above equations we develop a novel testing framework, DeepInspect, to systematically inspect a DNN implementing image classification tasks and look for potential confusion and bias related errors. We implemented DeepInspect in the Pytorch deep learning framework and Python 2.7. All of our experiments were run on Ubuntu 18.04.2 with two TITAN Xp GPUs.

IV. Experimental Design

A. Study Subjects

We apply DeepInspect for both multi-label and single-label DNN-based classifications. Under different settings, DeepInspect automatically inspects 10 DNN models for 8 datasets. Table II summarizes our study subjects. We used pre-trained models as shown in the table for all the settings except for COCO with gender. For COCO with gender model, we used the gender labels from [5] and trained the model in the same way as [5]. There are 11,538 entities and 1788 roles in total in the imSitu dataset. When inspecting imSitu model, we only considered the top 100 most frequent entities or roles in the test dataset.

Among the 10 DNN models, four of them are pre-trained robust models that are trained using adversarial images along with regular images. These models are pre-trained by provably robust training approach proposed in [39]. Three robust models with different network structures are trained using the CIFAR-10 dataset [39]. The last robust model is a pre-trained model in the Tiny ImageNet dataset [42].

B. Identifying Ground Truth (GT) Errors

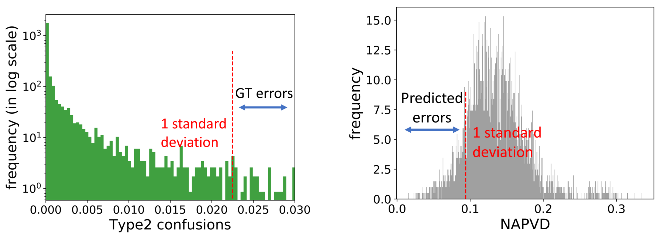
To collect the ground truth we refer to the test images *truly* misclassified by a given model. We then aggregate these misclassified image points by their real and predicted class-labels and estimate pair-wise confusion/bias.

1) GT of Confusion Errors: Confusion occurs when a DNN often makes mistakes in disambiguating members of two different classes. In particular, if a DNN is confused between two classes, the classification error rate is higher between the two classes than the rest of the class-pairs. Based on this, we define two types of confusion errors for single-label classification and multi-label classification separately:

Type1 confusions: In single-label classification, Type1 confusion occurs when an object of true label a (*e.g.*,`violin`) is

TABLE II: **Study Subjects**

Dataset			Model			
Task	**Name**	**#classes**	**CNN Models**	**#Neurons**	**#Layers**	**Reported Result**
Multi-label classification	COCO [23]	80	ResNet-50[5]	26,560	53 Conv	0.73 mean average precision
	COCO with gender[5]	81	ResNet-50[2]	26,560	53 Conv	0.71 mean average precision
	NUS[24]	1000	ResNet-18[37]	5,800	21 Conv	0.26 mean average precision
	imSitu[25]	205,095	ResNet-34[25]	8,448	36 Conv	0.37 mean accuracy
Single-label classification	CIFAR-100[21]	100	CNN[38]	2,916	26	0.74 accuracy
	Robust CIFAR-10[21] (R CIFAR-10)	10	Small (S) CNN[39]	158	8	0.69 accuracy
			Large (L) CNN[39]	1,226	14	0.73 accuracy
			ResNet[39] (R)	1,410	34	0.70 accuracy
	ImageNet[22]	1000	ResNet-50[40]	26,560	53 Conv	0.75 accuracy
	Robust Tiny ImageNet[41]	200	ResNet[42]	1,410	34	0.27 accuracy



(a) Confusions distribution (b) NAPVD distribution

Fig. 3: **Identifying Type2 confusions for multi-classification applications.** LHS shows how we marked the ground truth errors based on Type2 confusion score. RHS shows DeepInspect's predicted errors based on NAPVD score.

misclassified to another class b (*e.g.,*`cello`). For all the objects of class a and b, it can be quantified as: $\mathrm{type1conf}(a, b) = \mathrm{mean}(\mathrm{P}(a|b), \mathrm{P}(b|a))$—DNN's probability to misclassify class b as a and vice-versa, and takes the average value between the two. For example, given two classes `cello` and `violin`, type1conf estimates the mean probability of `violin` misclassified to `cello` and vise-versa. Note that, this is a bi-directional score, *i.e.* misclassification of b as a is the same as misclassification of a as b.

Type2 confusions: For multi-label classification, Type2 confusion occurs when an input image contains an object of class a (*e.g.,*`keyboard`) and no object of class b (*e.g.,*`mouse`), but the model predicts both classes (See Figure 8. For a pair of classes, this can be quantified as: $\mathrm{type2conf}(a, b) = \mathrm{mean}(\mathrm{P}((a, b)|a), \mathrm{P}((a, b)|b))$ to compute the probability to detect two objects in the presence of only one. For example, given two classes `keyboard` and `mouse`, type2conf estimates the mean probability of `mouse` being predicted while predicting `keyboard` and vise-versa. Similar to Type1 confusion, this is also a bi-directional score.

We measure type1conf and type2conf by using a DNN's *true classification error* measured on a set of test images. They create the DNN's true confusion characteristics between all possible class-pairs. We then draw the distributions of type1conf and type2conf, as shown in Figure 3a. The class-pairs having confusion scores greater than 1 standard deviation from the mean-value are then marked as pairs truly confused by the model and form our ground truth of confusion errors. For example, in COCO dataset, there are 80 classes and thus 3160 class pairs(80*79/2). 178 class-pairs of them are ground-truth confusion errors.

Note that, different from how a bug/error is defined in traditional software engineering, our suspicious confusion pairs have an inherent probabilistic nature. For example, even if a and b represent a confusion pair, it does not mean that all the images containing a or b will be misclassified by the model. Rather, it means that compared with other pairs, images containing a or b tend to have higher chance to be misclassified by the model.

2) GT of Bias: A DNN model is *biased* if it treats two classes differently. For example, consider three classes: `man`, `woman`, and `surfboard`. An unbiased model should not have different error rates while classifying `man` or `woman` in the presence of `surfboard`. To measure such bias formally, we define **confusion disparity** (cd) to measure differences of error between classes a and c and that between b and c: $\mathrm{cd}(a, b, c) = |error(a, c) - error(b, c)|$, where the error measure can be either type1conf or type2conf as defined earlier. cd essentially estimates the disparity of the model's error between classes a, b (*e.g.,* `man`, `woman`) *w.r.t.* a third class c (*e.g.,* `surfboard`).

We also define an aggregated measure **average confusion disparity** (abbreviated as **avg_cd**) between two classes a and b by summing up the bias between them over all third classes and take average:

$$\mathrm{avg\_cd}(a, b) := \frac{1}{|O| - 2} \sum_{c \in O, c \neq a, b} \mathrm{cd}(a, b, c).$$

Depending on the error types we used to estimate avg_cd, we refer to $Type1$_avg_cd and $Type2$_avg_cd. We measure avg_cd using true classification error rate reported by DNN for the test images. Similar to confusion errors, we draw the distribution of avg_cd for all possible class pairs and then consider the pairs as *truly biased* if their avg_cd score is higher than 1 standard deviation from the mean value. Such truly biased pairs form our ground truth of bias errors.

C. Evaluating DeepInspect

We evaluate DeepInspect using test set.

1) DeepInspect's Error Reporting: DeepInspect reports confusion errors based on NAPVD (See Equation (2)) scores—lower NAPVD indicates errors. We draw the distributions of NAPVDs for all possible class pairs, as shown in Figure 3b. Class pairs having NAPVD scores lower than 1 standard deviation from the mean score is marked as potential confusion errors.

As discussed in Section III-C, DeepInspect reports bias errors based on avg_bias score (See Equation (4)), where higher avg_bias means class pairs are more prone to bias errors. Similar to above, from the distribution of avg_bias scores, DeepInspect predicts pairs with avg_bias greater than 1 standard deviation from the mean score to be erroneous. Note that, while calculating error disparity between classes a, b *w.r.t.* c (See Equation (3)), if both a and b are far from c in the embedded space ρ, disparity of their distances (Δ) should not reflect true bias. Thus, while calculating avg_bias(a, b) we further filter out the triplets where $\Delta(c, a) > th \wedge \Delta(c, b) > th$, where th is some pre-defined threshold. In our experiment, we remove all the class-pairs having Δ larger than 1 standard deviation (*i.e.* th) below the mean value of all $Delta$s across all the class-pairs.

2) Evaluation Metric: We evaluate DeepInspect in two ways:

Precision & Recall: We use precision and recall to measure DeepInspect's accuracy. For each error type t, suppose that E is the number of errors detected by DeepInspect and A is the number of true errors in the ground truth set. Then the precision and recall of DeepInspect are $\frac{|A \cap E|}{|E|}$ and $\frac{|A \cap E|}{|A|}$ respectively.

Area Under Cost Effective Curve (AUCEC): Similar to how static analysis warnings are ranked based on their priority levels [43], we also rank the erroneous class-pairs identified by DeepInspect based on the decreasing order of error proneness, *i.e.* most error-prone pairs will be at the top. To evaluate such ranking we use a cost-effectiveness measure [44], AUCEC (Area Under the Cost-Effectiveness Curve), which has become standard to evaluate rank-based bug-prediction systems [43], [45], [46].

Cost-Effectiveness evaluates when we inspect/test top n% class-pairs in the ranked list (*i.e.* inspection cost), how many true errors are found (*i.e.* effectiveness). Both cost and effectiveness are normalized to 100%. Figure 7 shows cost on the x-axis, and effectiveness on the y-axis indicating the portion of the ground truth errors found. AUCEC is the area under this curve. We compare DeepInspect's performance against a random model that picks random class-pairs for inspection [47]. We also show the performance of an optimal model that ranks the class-pairs perfectly—if $n\%$ of all the class-pairs are truly erroneous, the optimal model would rank them at the top such that with lower inspection budget most of the errors will be detected. The optimal curve gives the upper bound of the ranking scheme.

V. Results

We begin our investigation by checking whether de-facto neuron coverage based metrics can capture class separation.

RQ1. For image classification applications, can white-box metrics efficiently distinguish between different classes?

Here we first investigate whether popular white-box metrics can distinguish between different classes. Then we investigate whether DeepInspect can capture these differences. We evaluate

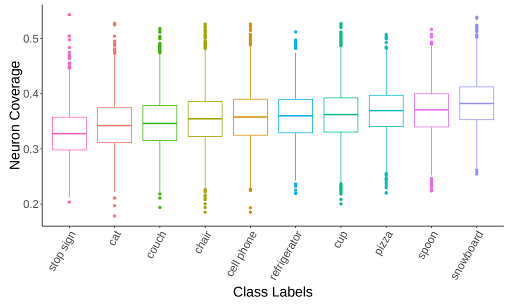


Fig. 4: **Distribution of neuron coverage per class-label for COCO dataset. For clarity, the coverage is shown for randomly picked 10 labels.**

this RQ *w.r.t.* the training data since the DNN behaviors are not tainted with inaccuracies associated with the test images.

RQ1a. Can Neuron Coverage distinguish between different classes? *Neuron Coverage* (NC), proposed by Pei *et al.* [14], computes the ratio of unique neurons activated by an input set and the total number of neurons in a DNN. Here we compute NC per class-label, *i.e.* for a given class-label, we measure the number of neurons activated by the images tagged with that label *w.r.t.* the total neurons. The activation threshold we use is 0.5, which is the same as used by Pei *et al.* [14]. We perform this experiment on COCO and CIFAR-100 to study multi- and single-label classifications. Figure 4 shows results for COCO. We observe similar results for CIFAR-100 .

Each boxplot in the figure shows the distribution of neuron coverage per class-label across all the relevant images. These boxplots visually show that *different labels* have very *similar* NC distribution. We further compare these distributions using Kruskal Test [48], which is a non-parametric way of comparing more than two groups. Note that we choose a non-parametric measure as NCs may not follow normal distributions. (Kruskal Test is a parametric equivalent of the one-way analysis of variance (ANOVA)) The result reports a $p-value << 0.05$, *i.e.* some differences exist across these distributions. However, a pairwise Cohend's effect size for each class-label pair, as shown in the following table, shows more than 56% and 78% class-pairs for CIFAR-100 and COCO have small to negligible effect size. This means neuron coverage cannot reliably distinguish a majority of the class-labels.

Effect Size of neuron coverage across different classes

Exp Setting	negligible	small	medium	large
COCO	40.51%	38.19%	16.96%	4.34%
CIFAR-100	31.94%	25.69%	23.87%	18.48%

RQ1b. Can DeepGauge [30] distinguish between different classes? Ma *et al.* argue that each neuron has a primary region of operation; they identify this region by using a boundary condition $[low, high]$ on its output during the training time; outputs outside this region $((-\infty, low) \cup (high, +\infty))$ are marked as corner cases. Leveraging this notion, they introduce multi-granular neuron and layer-level coverage criteria. For neuron coverage they propose: (i) *k-multisection coverage* to evaluate how thoroughly the primary region of a neuron is covered, (ii) *boundary coverage* to compute how many

corner cases are covered, and (iii) *strong neuron activation coverage* to measure how many corner case region is covered in $(high, +\infty)$ region. For layer-level coverage, they define (iv) *top-k neuron coverage* to identify the most active k-neurons for each layer, and (v) *top-k neuron pattern* for each test-case to find a sequence of neurons from the top-k most active neurons across each layer.

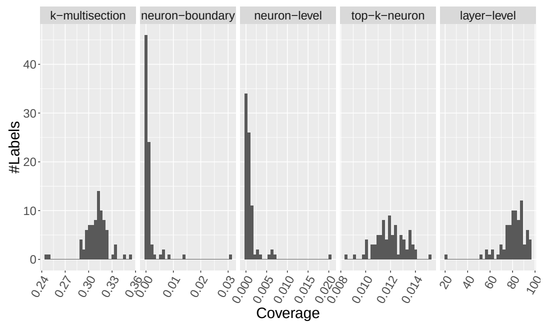


Fig. 5: **Histogram of DeepGauge multi-granular coverage per class-labels for COCO dataset**

We investigate whether each of these metrics can distinguish between different classes by measuring the above metrics for individual input classes following the original paper methodology. We first profiled every neuron upper- and lower-bound for each class using the training images containing that class-label. Next, we computed per class neuron coverages using test images containing that class; for k-multisection coverage we chose $k = 100$. For layer level coverages, we directly used the input images containing each class, where we select $k = 1$.

Figure 5 shows the results, *i.e.* histogram of the above five coverage criteria for COCO dataset. For all the five coverage criteria, there are many class-labels that share similar coverage. For example, in COCO , there are 52 labels with k-multisection neuron coverage with values between 0.31 and 0.32 (see Figure 5). Similarly, there are 40 labels with 0 neuron boundary coverage. Therefore, none of the five coverage criteria is an effective way to distinguish between different classes. Similar conclusion is drawn for CIFAR-100 dataset.

RQ1c. Can DeepInspect distinguish between different classes? Our white box metric, Neuron Activation Probability Matrix (ρ), by construction is designed per class. Hence it will be unfair to directly measure its capability to distinguish between different classes. Thus, we pose this question in a slightly different way, as described below. For multi-label classification, each image contains multiple class-labels. For example, an image can have labels for both `mouse` and `keyboard`. Such coincidence of labels may create confusion—if two labels always appear together in the ground truth set, no classifier can distinguish between them. To check how many times two labels coincide, we define a coincidence score between two labels L_a and L_b as: $coincidence\,(L_a, L_b) = mean(P\,(L_a, L_b|L_a)\,, P\,(L_a, L_b|L_b))$.

The above formula computes the minimum probability of labels L_a and L_b occurring together in an image given that one of them is present. Note that this is a bi-directional score, *i.e.* we treat the two labels similarly. The $mean$ operation ensures we detect the average coincidence of two directions. A low value of coincidence_score indicates two class-labels are easy to separate and vise-versa.

Now, to check DeepInspect's capability to capture class separation, we simply check the correlation between coincidence_score and confusion score (NAPVD) from Equation 2 for all possible class-label pairs. Since only multi-label objects can have label coincidences, we perform this experiment for a pre-trained ResNet-50 model on the COCO multi-label classification task. A Spearman correlation coefficient between the confusion and coincidence scores reaches a value as high as 0.96, showing strong statistical significance. The result indicates that DeepInspect can disambiguate most of the classes that have a low confusion score.

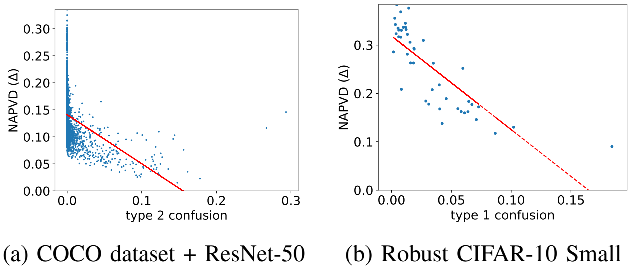
Interestingly, we found some pairs where coincidence score is high, but DeepInspect was able to isolate them. For example, (`cup`, `chair`), (`toilet`, `sink`) etc. Manually investigating such cases reveals that although these pairs often appear together in the input images, there are also enough instances when they appear by themselves. Thus DeepInspect disambiguates between these classes and puts them far in the embedded space ρ. These results indicate DeepInspect can also learn some hidden patterns from the context and thus, can go beyond inspecting the training data coincidence for evaluating model bias/confusion, which is the de-facto technique among machine learning researchers [5].

Result 1: *DeepInspect is a better choice for identifying class-separations than existing coverage based metrics for image classification task.*

We now investigate DeepInspect's capability in detecting confusion and bias errors in DNN models.

RQ2. Can we use DeepInspect to identify confusion errors of a DNN model?

In this RQ, we report DeepInspect's ability to detect Type1/Type2 confusions *w.r.t.* to ground truth confusion errors, as described in Section IV-B1.



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

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

We first explore the correlation between NAPVD and ground truth Type1/Type2 confusion score. Strong correlation has been found for all the 10 experimental settings. Figure 6 gives examples on COCO and CIFAR-10. These results indicate that

TABLE III: **Performance of DeepInspect on detecting confusion errors at two different settings.**

		NAPVD < mean-1std				Top 1%			
		TP	**FP**	**Precision**	**Recall**	**TP**	**FP**	**Precision**	**Recall**
COCO	DeepInspect	138	256	0.35	0.775	31	0	1	0.174
	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	0.19
	random	22	403	0.052	0.131	1	31	0.031	0.006
NUS	DeepInspect	2132	66453	0.031	0.869	1209	3780	0.242	0.493
	random	337	68248	0.005	0.137	24	4965	0.005	0.01
CIFAR-100	DeepInspect	206	584	0.261	0.718	39	10	0.796	0.136
	random	45	745	0.057	0.157	2	47	0.041	0.007
R CIFAR-10 S	DeepInspect	4	6	0.4	0.8	-	-	-	-
	random	1	9	0.1	0.2	-	-	-	-
R CIFAR-10 L	DeepInspect	3	4	0.43	0.6	-	-	-	-
	random	0	7	0	0	-	-	-	-
R CIFAR-10 R	DeepInspect	5	3	0.625	1	-	-	-	-
	random	0	8	0	0	-	-	-	-
Tiny ImageNet	DeepInspect	832	2477	0.251	0.607	122	77	0.613	0.089
	random	227	3082	0.069	0.166	13	185	0.065	0.01
ImageNet	DeepInspect	4014	69957	0.054	0.617	1073	3922	0.215	0.165
	random	962	73009	0.013	0.148	65	4930	0.013	0.01
imSitu	DeepInspect	48	58	0.453	0.165	31	19	0.62	0.107
	random	6	100	0.057	0.02	2	48	0.04	0.007

NAPVD can be used to detect confusion errors—lower NAPVD means more confusion.

By default, DeepInspect reports all the class-pairs with NAPVD scores 1 standard deviation less than mean NAPVD score as error-prone (See Figure 3b). In this setting, as the result shown on Table III, DeepInspect reports errors at high recall under most settings. Specifically, on NUS, CIFAR-100 and robust CIFAR-10 ResNet, DeepInspect can report errors as high as 86.9%, 71.8%, and 100%, respectively. In total, DeepInspect has identified thousands of confusion errors.

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 III, when the top1% confusion errors are reported, a much higher precision have been achieved for all the datasets. In particular, DeepInspect identifies 31 and 39 confusion errors for COCO model and CIFAR-100 model with 100% and 79.6% precisions respectively. The trade-off between precision and recall can be found on the cost-effective curves shown on Figure 7, which shows overall performance of DeepInspect at different inspection cutoffs. Overall, *w.r.t.* a random baseline model, DeepInspect is gaining AUCEC performance from 61.6% to 85.7%. In fact, for the Robust CIFAR-10 models, DeepInspect's performance is close to optimal.

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

Across all the three robust CIFAR-10 models, DeepInspect identifies (cat, dog), (bird, deer) and (automobile, truck) as erroneous pairs where one class is very likely to be mistakenly classified as the other class of the pair. This indicates that these confusion errors are to be tied to the training data, so all the models trained in this dataset including robust models may have these errors. These results further show that the confusion errors are orthogonal to the norm-based adversarial perturbations and we need a different technique to address them.

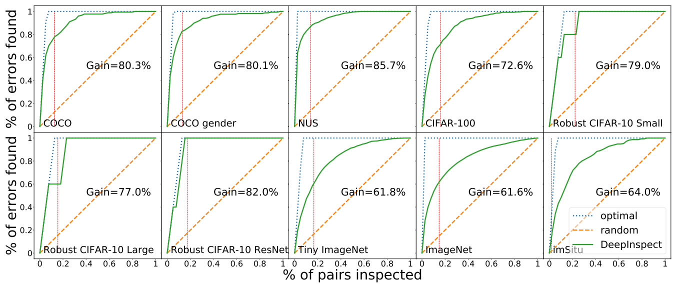
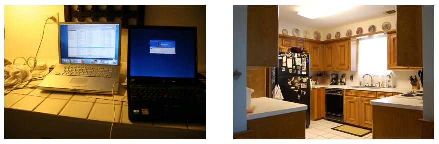
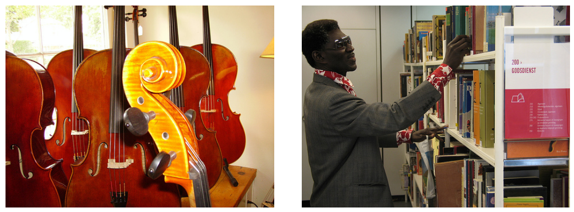


Fig. 7: **AUCEC plot of Type1/Type2 Confusion errors detected in 10 different settings.** The red vertical line shows the mark of 1-standard deviation less from mean NAVD score. DeepInspect marks all class-pairs with NAVD scores less than the red mark as potential errors. The gain reports DeepInspect's AUCEC gain over random.



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

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



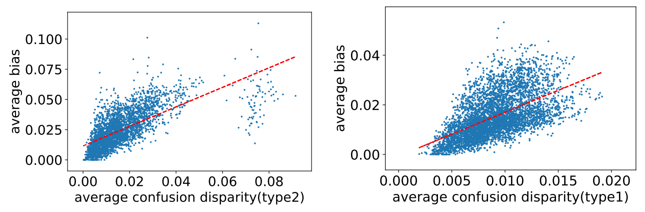
(a) (cello, violin) (b) (library, bookshop)

Fig. 9: **Confusion errors identified in ImageNet model. In each pair the second object is mistakenly identified by the model.**

Result 2: *DeepInspect can find confusion errors with precision 21% to 100% at top1% for both single- and multi-object classification tasks. DeepInspect also detects confusion errors in state-of-the art robust models.*

RQ3. Can we use DeepInspect to identify bias errors of a DNN model?

We evaluate this RQ by estimating a model's bias (avg_bias) using Equation (4) *w.r.t.* the ground truth (avg_cd) computed following Section IV-B2. We first explore the correlation between pairwise avg_cd and our proposed pairwise avg_bias; Figure 10 shows the results for the COCO dataset and the



(a) COCO (b) CIFAR-100

Fig. 10: **Strong positive Spearman's correlation (0.76 and 0.62) exists between avg_cd and avg_bias while detecting classification bias.**

CIFAR-100 dataset. Similar trends also happen in other datasets we study. The results show that a strong correlation exists between avg_cd and avg_bias. In other words, our proposed avg_bias is a good proxy for detecting bias errors.

TABLE IV: **Performance of DeepInspect on detecting bias errors at two different cutoffs**

		NAPVD > 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
	random	54	472	0.103	0.167	3	28	0.103	0.01
COCO gender	DeepInspect	218	325	0.401	0.568	17	16	0.515	0.044
	random	64	478	0.118	0.168	3	28	0.118	0.01
NUS	DeepInspect	23275	25662	0.476	0.44	1900	3095	0.38	0.036
	random	5180	43756	0.106	0.098	528	4466	0.106	0.01
CIFAR-100	DeepInspect	310	543	0.363	0.38	29	21	0.58	0.036
	random	140	711	0.165	0.172	8	41	0.165	0.01
R CIFAR-10 S	DeepInspect	7	4	0.636	0.778	-	-	-	-
	random	2	8	0.2	0.222	-	-	-	-
R CIFAR-10 L	DeepInspect	6	7	0.462	0.667	-	-	-	-
	random	2	9	0.2	0.267	-	-	-	-
R CIFAR-10 R	DeepInspect	6	3	0.667	0.667	-	-	-	-
	random	1	7	0.2	0.2	-	-	-	-
Tiny ImageNet	DeepInspect	1361	1859	0.423	0.453	142	57	0.714	0.047
	random	486	2732	0.151	0.162	30	168	0.151	0.01
ImageNet	DeepInspect	26704	48913	0.353	0.33	3253	1742	0.651	0.04
	random	12234	63381	0.162	0.151	808	4186	0.162	0.01
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	0.01

As in RQ2, we also do a precision-recall analysis of finding the bias errors across all the datasets at two specific cutoffs. We analyze the precision and recall of DeepInspect when reporting bias errors at cutoff Top1%(avg_bias) and mean(avg_bias)+standard deviation(avg_bias), respectively. The results are shown in Table IV. At cutoff Top1%(avg_bias), DeepInspect has high precision. In particular, DeepInspect can detect ground truth 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 relatively low precision. In particular, DeepInspect can detect ground truth suspicious pairs with recall as high as 75.9% and 71.8% recall for COCO and imSitu respectively. DeepInspect can report 657(=249+408) true bias errors in total for the two models. DeepInspect outperforms the random baseline by a large margin at both cutoffs. As in the case of detecting confusion errors, a trade-off between precision and recall exists here. This can be customized based on a user's need. The cost-effective analysis, as shown on Figure 11, shows the entire spectrum.

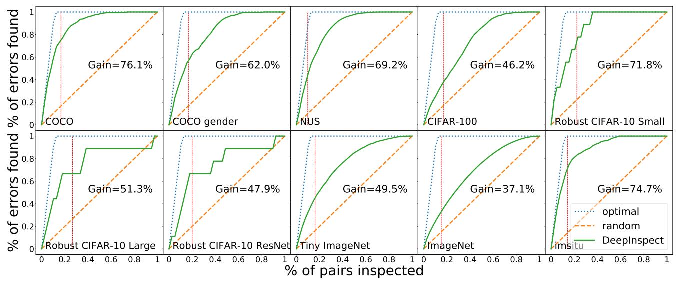


Fig. 11: **Bias errors detected *w.r.t.* the ground truth of avg_cd beyond one standard deviation from mean.** The gain reports DeepInspect's AUCEC gain over random.

As shown in the figure, DeepInspect outperforms the random baseline by a large margin. The AUCEC gains of DeepInspect over the random baseline are from 37% to 76% across the 10 settings. The performance of DeepInspect 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.

Inspired by [5], which shows bias exists between man and woman in COCO with gender in the task of image captioning, we analyze the most biased third class c for a and b being man and woman in COCO and imSitu. As shown in Figure 12, we found that sports like skis, snowboard, and surfboard is more closely associated with man and thus misleads the model to predict women in the images to be men. Figure 13 shows results on imSitu, we found that the model tends to associate the class "inside" with woman while associate the class "outside" with man.

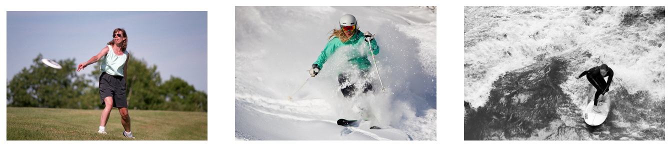


Fig. 12: **The model classifies the women in these pictures as men in COCO dataset.**

We generalize the idea by choosing classes a and b to be any class-pair. We found that similar bias also exists in the single-label classification settings. For example, in ImageNet, one of the highest bias is between Eskimo_dog and rapeseed *w.r.t.* Siberian_husky. The model tends to confuse between the two dogs but not between Eskimo_dog and rapeseed. This makes sense since Eskimo_dog and Siberian_husk are all dogs and are more easily to be misclassified by the model.

Note that, one of the fairness violations of a DNN system is the drastic difference on accuracy across groups divided according to some sensitive feature(s). In black-box testing, the tester can get a number indicating the degree of fairness has been violated by feeding into the model a validation set. In contrast, DeepInspect provides a new angle to the fairness violations. The neuron distance difference between two classes

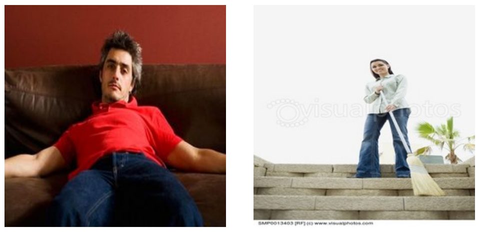


Fig. 13: **The model classifies the man in the first figure to be a woman and the woman in the second figure to be a man.**

a and b *w.r.t.* a third class c sheds light on why the model tends to be more likely to confuse between one of them and c than the other. We leave a more comprehensive examination on interpreting bias/fairness violations for future work.

Result 3: *DeepInspect can find bias errors for both single- and multi-label classification tasks and even for the robust models from 38% to 84% precisions at top 1%.*

VI. Threats to Validity

Although DeepInspect can find confusion and bias errors, its performance relies on the the accuracy of the model it tests. Since DeepInspect groups images according to the predicted labels, in the case when the model has very low testing accuracy, the embedding of objects in ρ will not be accurate and thus leads to inferior testing performance.

Another limitation is that DeepInspect needs to decide thresholds for both confusion errors and bias errors. We minimize this threat by choosing thresholds that are 1 standard deviation far from the corresponding mean values.

The task of classifying any possible object accurately is notoriously difficult. Here we simplify the problem to test the DNN model for the classes that it has seen before during the training process. For example, if the model is trained with `cow`'s images, we will test the model with variations of `cows` correctly. However, we will not test the model for `dinosaur` if it has never seen it during the training process.

VII. Related Work

Software Testing & Verification of DNN. Prior research proposed different white-box testing criteria based on neuron coverage [13], [14], [30] and neuron-pair coverage [31]. Sun *et al.* [49] further presented a concolic testing approach for DNNs. They showed that their concolic testing approach can effectively increase coverage and find adversarial examples. There are also efforts to verify DNNs [35], [50]–[52] against adversarial attacks. However, most of the verification efforts are limited to small DNNs and limited system-wide properties (*e.g.,* range of pixel values). In contrast, we propose testing strategies that test class separations. Our initial results indicate that such metrics have potential to identify potential bias and weakness of end-to-end DNN applications.

Adversarial Deep Learning. DNNs are known to be vulnerable to well-crafted inputs called adversarial examples, which are imperceptible to a human but can easily make DNNs fail [53]–[60]. Much work has been done to defend the adversarial attacks [61]–[69]. Our methods has potential to identify adversarial inputs. Moreover, adversarial examples are usually out of distribution data and not realistic, while we do not require to generate any new transformed images to find errors. Further, we can identify a general weakness or errors rather than focusing on crafted attacks that often require strong attacker model.

Interpreting DNN. There are many research on model interpretability and visualization [70]–[75]. In particular, Dong *et al.* [75] observed that instead of learning the semantic features of the whole objects, neurons tend to react on different parts of the objects in a recurrent manner. Our probabilistic way to look at neuron activation per class aims to capture a wholistic behavior of an entire classes instead of individual object so that diverse features of class members can be captured. Closest to ours is by Papernot *et al.* [76] who used nearest training points to explain the adversarial attacks. In comparison, we analyze the DNN's dependencies on an entire training/testing data and represent it in a matrix. By inspecting this matrix, we can explain the bias and weaknesses of the DNN.

Evaluating model's Bias/Fairness. Evaluating bias and fairness of a system is important both from a theoretical and a practical perspective [77]–[80]. At a high level, the related studies first define a fairness criteria and then try to optimize the original objective while satisfying the fairness creteria [81]–[86]. These properties are defined either at individual [81], [87], [88] or group levels [82], [89], [90]. In this work, we showed the potential of DeepInspect in detecting group-level fairness.

Galhotra *et al.* [91] first applied the notion of software testing for evaluating software fairness. In particular, 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 it tests takes into sensitive attribute(s) during training and inference time. Such 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 work, we use a white-box approach to measure the bias learned by the model during training. Our testing method does not assume the model we are testing taking into 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 it for any deep learning models even when only unlabeled testing data is provided. In addition, our method tries to provide an explanation behind such discrimination. A complementary approach by Papernot *et al.* [76] shows such explainability behind model bias in single-label classification setting.

VIII. Conclusion

In this paper, we propose a white box DNN testing framework, which can automatically detect confusion and bias errors in DNN-based image classification models. We implemented DeepInspect and applied it on 8 different popular image classification datasets and 10 pretrained DNN models including 4 pre-trained robust models. We show that DeepInspect is able to detect errors for both single- and multi-label classification models with high precision. We also put all the errors that DeepInspect detected on the public website.

In this work, we mainly focus on detecting confusion/bias errors. A natural follow-up question is how to fix these errors. Unlike fixing bugs in traditional software, fixing errors in DNNs is an open problem and often requires retraining the models. Preliminary results on COCO dataset have shown that by augmenting the training datasets with images from confusing class pairs could reduce the confusion errors. We leave a more comprehensive examination of how to fix the confusion and bias errors found by DeepInspect to the future work.

References

[1] P. Kamavisdar, S. Saluja, and S. Agrawal, "A survey on image classification approaches and techniques," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 2, no. 1, pp. 1005–1009, 2013.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[3] L. Grush, "Google engineer apologizes after photos app tags two black people as gorillas," 2015. [Online]. Available: https://www.theverge.com/2015/7/1/8880363/google-apologizes-photos-app-tags-two-black-people-gorillas

[4] MalletsDarker, "I took a few shots at lake louise today and google offered me this panorama," 2018. [Online]. Available: https://www.reddit.com/r/funny/comments/7r9ptc/i_took_a_few_shots_at_lake_louise_today_and/dsvv1nw/

[5] J. Zhao, T. Wang, M. Yatskar, V. Ordonez, and K.-W. 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*, 2017, pp. 2941–2951. [Online]. Available: https://www.aclweb.org/anthology/D17-1319

[6] A. Rosenfeld, R. S. Zemel, and J. K. Tsotsos, "The elephant in the room," *CoRR*, vol. abs/1808.03305, 2018. [Online]. Available: http://arxiv.org/abs/1808.03305

[7] A. Rose, "Are face-detection cameras racist?" 2010. [Online]. Available: http://content.time.com/time/business/article/0,8599,1954643,00.html

[8] J. Buolamwini and T. Gebru, "Gender shades: Intersectional accuracy disparities in commercial gender classification," in *FAT*, 2018.

[9] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. S. Openai, D. Mané, and G. Brain, "Concrete Problems in AI Safety." [Online]. Available: https://arxiv.org/pdf/1606.06565.pdfhttp://arxiv.org/abs/1606.06565.pdf

[10] B. J. Taylor and M. A. Darrah, "Rule extraction as a formal method for the verification and validation of neural networks," in *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, vol. 5. IEEE, 2005, pp. 2915–2920.

[11] S. A. Seshia, D. Sadigh, and S. S. Sastry, "Towards verified artificial intelligence," *arXiv preprint arXiv:1606.08514*, 2016.

[12] T. Dreossi, S. Jha, and S. A. Seshia, "Semantic adversarial deep learning," in *International Conference on Computer-Aided Verification (CAV)*, 2018.

[13] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *International Conference of Software Engineering (ICSE), 2018 IEEE conference on*. IEEE, 2018.

[14] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," pp. 1–18, 2017. [Online]. Available: http://doi.acm.org/10.1145/3132747.3132785

[15] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[16] T. M. Mitchell, *Machine Learning*, 1st ed. New York, NY, USA: McGraw-Hill, Inc., 1997.

[17] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[18] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[19] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, and M. Young, "Machine learning: The high interest credit card of technical debt," 2014.

[20] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," *International Journal of Data Warehousing and Mining (IJDWM)*, vol. 3, no. 3, pp. 1–13, 2007.

[21] A. Krizhevsky, "Learning multiple layers of features from tiny images," *University of Toronto*, 05 2012.

[22] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[23] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[24] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y.-T. Zheng, "Nus-wide: A real-world web image database from national university of singapore," in *Proc. of ACM Conf. on Image and Video Retrieval (CIVR'09)*, Santorini, Greece., July 8-10, 2009.

[25] M. Yatskar, L. Zettlemoyer, and A. Farhadi, "Situation recognition: Visual semantic role labeling for image understanding," in *Conference on Computer Vision and Pattern Recognition*, 2016.

[26] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[27] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[28] "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.

[29] "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.

[30] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu, J. Zhao, and Y. Wang, "Deepgauge: Multi-granularity testing criteria for deep learning systems," pp. 120–131, 2018. [Online]. Available: http://doi.acm.org/10.1145/3238147.3238202

[31] Y. Sun, X. Huang, and D. Kroening, "Testing deep neural networks," *arXiv preprint arXiv:1803.04792*, 2018.

[32] I. Goodfellow and N. Papernot, "The challenge of verification and testing of machine learning," http://www.cleverhans.io/security/privacy/ml/2017/06/14/verification.html, 2017.

[33] T. J. Ostrand and M. J. Balcer, "The category-partition method for specifying and generating fuctional tests," *Communications of the ACM*, vol. 31, no. 6, pp. 676–686, 1988.

[34] D. Hamlet and R. Taylor, "Partition testing does not inspire confidence (program testing)," *IEEE Transactions on Software Engineering*, vol. 16, no. 12, pp. 1402–1411, 1990.

[35] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *International Conference on Computer Aided Verification*. Springer, 2017, pp. 3–29.

[36] J. Kim, R. Feldt, and S. Yoo, "Guiding deep learning system testing using surprise adequacy," in *Proceedings of the 41th International Conference on Software Engineering*, ser. ICSE 2019, 2019.

[37] V. O. Tianlu Wang, Kota Yamaguchi, "Deep residual learning for image recognition," in *Feedback-prop: Convolutional Neural Network Inference under Partial Evidence(CVPR)*, 2018.

[38] "Base pretrained models and datasets in pytorch," 2017. [Online]. Available: https://github.com/aaron-xichen/pytorch-playground

[39] E. Wong, F. Schmidt, J. H. Metzen, and J. Z. Kolter, "Scaling provable adversarial defenses," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 8410–8419. [Online]. Available: http://papers.nips.cc/paper/8060-scaling-provable-adversarial-defenses.pdf

[40] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations (ICLR)*, 2015.

[41] "Tiny imagenet visual recognition challenge," 2017. [Online]. Available: https://tiny-imagenet.herokuapp.com/

[42] S. Wang, Y. Chen, A. Abdou, and S. Jana, "Mixtrain: Scalable training of formally robust neural networks," *CoRR*, vol. abs/1811.02625, 2018. [Online]. Available: http://arxiv.org/abs/1811.02625

[43] F. Rahman and P. Devanbu, "How, and why, process metrics are better," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 432–441.
[44] E. Arisholm, L. C. Briand, and E. B. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models." *JSS*, vol. 83, no. 1, pp. 2–17, 2010.
[45] F. Rahman, D. Posnett, A. Hindle, E. Barr, and P. Devanbu, "Bugcache for inspections: hit or miss?" in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM, 2011, pp. 322–331.
[46] B. Ray, V. Hellendoorn, S. Godhane, Z. Tu, A. Bacchelli, and P. Devanbu, "On the" naturalness" of buggy code," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 2016, pp. 428–439.
[47] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.
[48] W. H. Kruskal and W. A. Wallis, "Use of ranks in one-criterion variance analysis," *Journal of the American Statistical Association*, vol. 47, no. 260, pp. 583–621, 1952. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/01621459.1952.10483441
[49] Y. Sun, M. Wu, W. Ruan, X. Huang, M. Kwiatkowska, and D. Kroening, "Concolic testing for deep neural networks," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE 2018. New York, NY, USA: ACM, 2018, pp. 109–119. [Online]. Available: http://doi.acm.org/10.1145/3238147.3238172
[50] K. Pei, Y. Cao, J. Yang, and S. Jana, "Towards practical verification of machine learning: The case of computer vision systems," *arXiv preprint arXiv:1712.01785*, 2017.
[51] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, *Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks*. Cham: Springer International Publishing, 2017, pp. 97–117.
[52] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Formal security analysis of neural networks using symbolic intervals," 2018.
[53] 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*, pp. 1–20, 2019.
[54] A. Raghunathan, J. Steinhardt, and P. Liang, "Certified defenses against adversarial examples," *6th International Conference on Learning Representations (ICLR)*, 2018.
[55] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *International Conference on Learning Representations (ICLR)*, 2015.
[56] A. Nguyen, J. Yosinski, and J. 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*, 2015, pp. 427–436.
[57] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2016, pp. 372–387.
[58] 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.
[59] S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel, "Adversarial attacks on neural network policies," *arXiv preprint arXiv:1702.02284*, 2017.
[60] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *Workshop track*, 2017.
[61] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. Nori, and A. Criminisi, "Measuring neural net robustness with constraints," in *Advances in Neural Information Processing Systems*, 2016, pp. 2613–2621.
[62] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 2017, pp. 39–57.
[63] S. Gu and L. Rigazio, "Towards deep neural network architectures robust to adversarial examples," in *International Conference on Learning Representations (ICLR)*, 2015.
[64] J. H. Metzen, T. Genewein, V. Fischer, and B. Bischoff, "On detecting adversarial perturbations," in *International Conference on Learning Representations (ICLR)*, 2017.
[65] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 2016, pp. 582–597.
[66] U. Shaham, Y. Yamada, and S. Negahban, "Understanding adversarial training: Increasing local stability of neural nets through robust optimization," *arXiv preprint arXiv:1511.05432*, 2015.
[67] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," *arXiv preprint arXiv:1704.01155*, 2017.
[68] S. Zheng, Y. Song, T. Leung, and I. Goodfellow, "Improving the robustness of deep neural networks via stability training," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4480–4488.
[69] W. He, J. Wei, X. Chen, N. Carlini, and D. Song, "Adversarial example defenses: Ensembles of weak defenses are not strong," in *Proceedings of the 11th USENIX Conference on Offensive Technologies*, ser. WOOT'17. Berkeley, CA, USA: USENIX Association, 2017, pp. 15–15. [Online]. Available: http://dl.acm.org/citation.cfm?id=3154768.3154783
[70] Z. C. Lipton, "The mythos of model interpretability," *Proceedings of the 33rd International Conference on Machine Learning Workshop*, 2016.
[71] Q.-s. Zhang and S.-C. Zhu, "Visual interpretability for deep learning: a survey," *Frontiers of Information Technology & Electronic Engineering*, vol. 19, no. 1, pp. 27–39, 2018.
[72] 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)*, Oct 2017, pp. 618–626.
[73] G. Montavon, W. Samek, and K.-R. Müller, "Methods for interpreting and understanding deep neural networks," *Digital Signal Processing*, 2017.
[74] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba, "Network dissection: Quantifying interpretability of deep visual representations," in *Computer Vision and Pattern Recognition*, 2017.
[75] Y. Dong, H. Su, J. Zhu, and F. Bao, "Towards interpretable deep neural networks by leveraging adversarial examples," *arXiv preprint arXiv:1708.05493*, 2017.
[76] N. Papernot and P. McDaniel, "Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning," *arXiv preprint arXiv:1803.04765*, 2018.
[77] B. T. Luong, S. Ruggieri, and F. 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*. ACM, 2011, pp. 502–510.
[78] R. Zemel, Y. Wu, K. Swersky, T. Pitassi, and C. Dwork, "Learning fair representations," in *Proceedings of the 30th International Conference on Machine Learning*, 2013, pp. 325–333.
[79] M. B. Zafar, I. Valera, M. Gomez Rodriguez, and K. P. Gummadi, "Fairness constraints: Mechanisms for fair classification," vol. 54, 2017.
[80] Y. Brun and A. 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*, ser. ESEC/FSE 2018. New York, NY, USA: ACM, 2018, pp. 754–759. [Online]. Available: http://doi.acm.org/10.1145/3236024.3264838
[81] C. Dwork, M. Hardt, T. Pitassi, O. Reingold, and R. S. Zemel, "Fairness through awareness," *In Proceedings of the Innovations in Theoretical Computer Science Conference*, vol. abs/1104.3913, pp. 214–226, 2012.
[82] M. Hardt, E. Price, and N. Srebro, "Equality of opportunity in supervised learning," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS'16, USA, 2016, pp. 3323–3331.
[83] S. Barocas, M. Hardt, and A. Narayanan, *Fairness and Machine Learning*. fairmlbook.org, 2018, http://www.fairmlbook.org.
[84] A. K. Menon and R. 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*, 2018, pp. 107–118. [Online]. Available: http://proceedings.mlr.press/v81/menon18a.html
[85] M. Donini, L. Oneto, S. Ben-David, J. Shawe-Taylor, and M. 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.*, 2018, pp. 2796–2806. [Online]. Available: http://papers.nips.cc/paper/7544-empirical-risk-minimization-under-fairness-constraints
[86] A. L. Lamy, Z. Zhong, A. K. Menon, and N. Verma, "Noise-tolerant fair classification," *CoRR*, vol. abs/1901.10837, 2019. [Online]. Available: http://arxiv.org/abs/1901.10837

[87] M. J. Kusner, J. Loftus, C. Russell, and R. Silva, “Counterfactual fairness,” in *Advances in Neural Information Processing Systems 30*, 2017, pp. 4066–4076.

[88] M. P. Kim, O. Reingold, and G. N. Rothblum, “Fairness through computationally-bounded awareness,” *32nd Conference on Neural Information Processing Systems (NeurIPS 2018)*, 2018.

[89] T. Calders, F. Kamiran, and M. Pechenizkiy, “Building classifiers with independency constraints,” in *2009 IEEE International Conference on Data Mining Workshops*, Dec 2009, pp. 13–18.

[90] M. B. Zafar, I. Valera, M. Gomez Rodriguez, and K. P. Gummadi, “Fairness beyond disparate treatment & disparate impact: Learning classification without disparate mistreatment,” in *Proceedings of the 26th International Conference on World Wide Web*, 2017, pp. 1171–1180.

[91] S. Galhotra, Y. Brun, and A. Meliou, “Fairness testing: testing software for discrimination,” in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM, 2017, pp. 498–510.