

Testing DNN Image Classifiers for Confusion & Bias Errors

Yuchi Tian*
Columbia University
yuchi.tian@columbia.edu

Ziyuan Zhong*
Columbia University
ziyuan.zhong@columbia.edu

Vicente Ordonez
University of Virginia
vicente@virginia.edu

Gail Kaiser
Columbia University
kaiser@cs.columbia.edu

Baishakhi Ray
Columbia University
rayb@cs.columbia.edu

ABSTRACT

Image classifiers are an important component of today’s software, from consumer and business applications to safety-critical domains. The advent of Deep Neural Networks (DNNs) is the key catalyst behind such wide-spread success. However, wide adoption comes with serious concerns about the robustness of software systems dependent on DNNs for image classification, as several severe erroneous behaviors have been reported under sensitive and critical circumstances. We argue that developers need to rigorously test their software’s image classifiers and delay deployment until acceptable. We present an approach to testing image classifier robustness based 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.

CCS CONCEPTS

• **Software and its engineering** → **Software testing and debugging**; • **Computing methodologies** → *Neural networks*.

KEYWORDS

whitebox testing, deep learning, DNNs, image classifiers, bias

1 INTRODUCTION

Image classification has a plethora of applications in software for safety-critical domains such as self-driving cars, medical diagnosis, *etc.* Even day-to-day consumer software includes image classifiers, such as Google Photo search and Facebook image tagging. Image classification is a well-studied problem in computer vision, where a model is trained to classify an image into single or multiple predefined categories [26]. Deep Neural Networks (DNNs) have enabled major breakthroughs in image classification tasks over the past few years, sometimes even matching human-level accuracy under some conditions [22], which has led to their ubiquity in modern software.

However, in spite of such spectacular success, DNN-based image classification models, like traditional software, are known to have serious bugs. For example, Google faced backlash in 2015 due to a notorious error in its photo-tagging app, which tagged pictures of dark-skinned people as “gorillas” [19]. Analogous to traditional software bugs, the Software Engineering (SE) literature denotes these classification errors as *model bugs* [43], which can arise due to either imperfect model structure or inadequate training data.

At a high-level, these bugs can affect either an *individual image*, where a particular image is mis-classified (*e.g.*, a particular skier is mistaken as a part of a mountain), or an *image class*, where a class of images is more likely to be mis-classified (*e.g.*, dark-skinned people are more likely to be misclassified), as shown in Table 1. The latter bugs are specific to a whole *class* of images rather than individual images, implying systematic bugs rather than the DNN equivalent of off-by-one errors. While much effort from the SE literature on Neural Network testing has focused on identifying individual-level violations—using white-box [29, 42, 60, 79], grey-box [43, 77], or concolic testing [75], detection of class-level violations remains relatively less explored. This paper focuses on automatically detecting such class-level bugs, so they can be fixed.

After manual investigation of some public reports describing the class-level violations listed in Table 1, we determined two root causes: (i) **Confusion**: The model cannot differentiate one class from another. For example, Google Photos confuses skier and mountain [44]. (ii) **Bias**: The model shows disparate outcomes between two related groups. For example, Zhao *et al.* in their paper “Men also like shopping” [92], find classification bias in favor of women on activities like shopping, cooking, washing, *etc.* We further notice that some class-level properties are violated in both kinds of cases. For example, in the case of *confusion errors*, the classification error-rate between the objects of two classes, say, skier and mountain, is significantly higher than the overall classification error rate of the model. Similarly, in the bias scenario reported by Zhao *et al.*, a

*Both are first authors, and contributed equally to this research.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ICSE '20, May 23–29, 2020, Seoul, Republic of Korea

© 2020 Association for Computing Machinery.

ACM ISBN 978-1-4503-7121-6/20/05...\$15.00

<https://doi.org/10.1145/3377811.3380400>

Table 1: Examples of real-world bugs reported in neural image classifiers

Bug Type	Name	Report Date	Outcome
Confusion	Gorilla Tag [19]	Jul 1, 2015	Black people were tagged as gorillas by Google photo app.
	Elephant is detected in a room [68]	Aug 9, 2018	Image Transplantation (replacing a sub-region of an image by another image containing a trained object) leads to mis-classification.
	Google Photo [44]	Dec 10, 2018	Google Photo confuses skier and mountain.
Bias	Nikon Camera [67]	Jan 22, 2010	Camera shows bias toward Caucasian faces when detecting people’s blinks.
	Men Like Shopping [92]	July 29, 2017	Multi-label object classification models show bias towards women on activities like shopping, cooking, washing, <i>etc.</i>
	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.

DNN model should not have different error rates while classifying the gender of a person in the shopping category. Unlike individual image properties, this is a class property affecting all the shopping images with men or women. Any violation of such a property by definition affects the whole class although not necessarily every image in that class, *e.g.*, a man is more prone to be predicted as a woman when he is shopping, even though some individual images of a man shopping may still be predicted correctly. Thus, we need a class-level approach to testing image classifier software for confusion and bias errors.

The bugs in a DNN model occur due to sub-optimal interactions between the model structure and the training data [43]. To capture such interactions, the literature has proposed various metrics primarily based on either neuron activations [29, 42, 60] or feature vectors [43, 53]. However, these techniques are primarily targeted at the individual image level. To detect class-level violations, we abstract away such model-data interactions at the class level and analyze the inter-class interactions using that new abstraction. To this end, we propose a metric using neuron activations and a baseline metric using weight vectors of the feature embedding to capture the class abstraction.

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 corresponds to a neuron activation probability. If the distance between the two vectors for two different classes is too close, compared to other class-vector pairs, that means the DNN under test may not effectively distinguish between those two classes. Motivated by MODE’s technique [43], we further create a baseline where each class is represented by the corresponding weight vector of the last linear layer of the model under test.

We evaluate our methodology for both single- and multi-label classification models in eight different settings. Our experiments demonstrate that DeepInspect can efficiently detect both Bias and Confusion errors in popular neural image classifiers. 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 [82]; DeepInspect finds hundreds of errors proving the need for orthogonal testing strategies to detect such class-level mispredictions. Unlike some other DNN testing techniques [53, 60, 75, 77], DeepInspect does not need to generate additional transformed (synthetic) images to find these errors. The primary contributions of this paper are:

- We propose a novel neuron-coverage metric to automatically detect class-level violations (confusion and bias errors) in DNN-based visual recognition models for image classification.
- We implemented our metric and underlying techniques in DeepInspect.
- We evaluated DeepInspect and found many errors in widely-used DNN models with precision up to 100% (avg. 72.6%) for confusion errors and up to 84.3% (avg. 66.8%) for bias errors.

Our code is available at <https://github.com/ARiSE-Lab/DeepInspect>. The errors reported by DeepInspect are available at: <https://www.ariselab.info/deepinspect>.

2 DNN BACKGROUND

Deep Neural Networks (DNNs) are a popular type of machine learning model loosely inspired by the neural networks of human brains. A DNN model learns the logic to perform a software task from a large set of *training examples*. For example, an image recognition model learns to recognize cows through being shown (trained with) many sample images of cows.

A typical "feed-forward" DNN consists of a set of connected computational units, referred as *neurons*, that are arranged sequentially in a series of *layers*. The neurons in sequential layers are connected to each other through *edges*. Each edge has a corresponding weight. Each neuron applies σ , a *nonlinear activation function* (*e.g.*, ReLU [50], Sigmoid [48]), to the incoming values on its input edges and sends the results on its output edges to the next layer of connected neurons. For image classification, convolutional neural networks (CNNs) [37], a specific type of DNN, are typically used. CNNs consist of layers with local spatial connectivity and sets of neurons with shared parameters.

When implementing a DNN application, developers typically start with a set of annotated experimental data and divide it into three sets: (i) training: to construct the DNN model in a supervised setting, meaning the training data is labeled (*e.g.*, using stochastic gradient descent with gradients computed using back-propagation [69]); (ii) validation: to tune the model’s hyper-parameters, basically configuration parameters that can be modified to better fit the expected application workload; and (iii) evaluation: to evaluate the accuracy of the trained model *w.r.t.* to its predictions on other annotated data, to determine whether or not it predicts correctly. Typically, training, validation, and testing data are drawn from the same initial dataset.

For image classification, a DNN can be trained in either of the following two 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 [78]. Among some popular image classification datasets, MNIST, CIFAR-10/CIFAR-100 [32] and ImageNet [70] 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[38] and imSitu[85] 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.

Given any single- or multi-label classification task, DNN classifier software 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 [6]. The DNN represents the input image in an embedded space with the feature vector at a certain intermediate layer and uses the layers after as a classifier to classify these representations. The *class separation* between two classes estimates how well the DNN has learned to separate each class from the other. If the embedded distance between two classes is 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.

3 METHODOLOGY

We give a detailed technical description of DeepInspect. We describe a typical scenario where we envision our tool might be used in the following and design the methodology accordingly.

Usage Scenario. Similar to customer testing of post-release software, 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. 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. As elaborated in Section 7, once these errors are reported back to the developers, they can focus their debugging and fixing effort on these classes. Figure 1 shows the DeepInspect workflow.

3.1 Definitions

Before we describe DeepInspect’s methodology in detail, we introduce definitions that we use in the rest of the paper. The following

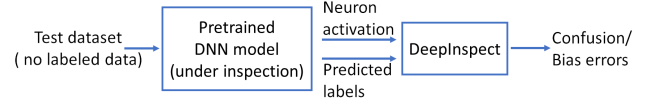


Figure 1: DeepInspect Workflow

table shows our notation.

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

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

Neural-Path per Class (NP_C). For a class C_i , this metric represents a set consisting of the union of neural-paths activated by all the inputs in C_i .

For example, consider a class cow containing two images: a brown cow and a black cow. Let’s assume 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 $(N_1^1, N_2^1, N_3^1, N_4^1, N_5^1)$, where the superscripts represent the number of times each neuron is activated by C_{cow} . Thus, each class C_i in a dataset can be expressed with a *neuron activation frequency vector*, which captures how the model interacts with C_i .

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

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 $P(N_j | C_i)$.

$$\rho = \begin{matrix} & C_1 & \dots & C_i & \dots & C_m \\ \begin{matrix} N_1 \\ \dots \\ N_j \\ \dots \\ N_n \end{matrix} & \begin{pmatrix} p_{11} & & & & p_{1m} \\ \dots & & & & \dots \\ p_{j1} & & \dots & & p_{jm} \\ \dots & & & & \dots \\ p_{n1} & & & & p_{nm} \end{pmatrix} \end{matrix} \quad (1)$$

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, C_s are predicted labels.

Since Neuron Activation Probability Matrix (ρ) is designed to represent each class, it should be able to distinguish between different C_s . Next, we use this metric to find two different classes of errors often found in DNN systems: *confusion* and *bias* (see Table 1).

3.2 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 1, 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 two classes but only one of them 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 euclidean distance between their two probability vectors:

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

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 Neuron Activation Probability Vector Distance).

3.3 Finding Bias Errors

In an object classification task, bias occurs if the model under test shows disparate outcomes between two related classes. 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 , $\Delta(\text{baby}, \text{woman})$ is much smaller than $\Delta(\text{baby}, \text{man})$. 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:

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

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:

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

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*.

Note that, even when the two classes a and b are not confused by the model, *i.e.* $\Delta(a, b) > \text{conf_th}$, they can still show bias *w.r.t.* another class, say c , if $\Delta(a, c)$ is very different from $\Delta(b, c)$. Thus, bias and confusion are two separate types of class-level errors that we intend to study in this work.

Table 2: Study Subjects

Classification Task	Dataset		Model			Reported Accuracy
	Name	#classes	CNN Models	#Neurons	#Layers	
Multi-label classification	COCO [38]	80	ResNet-50[92]	26,560	53 Conv	0.73*
	COCO gender[92]	81	ResNet-50[22]	26,560	53 Conv	0.71*
	imSitu[85]	205,095	ResNet-34[85]	8,448	36 Conv	0.37†
Single-label classification	CIFAR-100[32]	100	CNN[1]	2,916	26	0.74
	Robust	10	Small CNN[82]	158	8	0.69
	CIFAR-10[32]		Large CNN[82]	1,226	14	0.73
			ResNet[82]	1,410	34	0.70
	ImageNet[70]	1000	ResNet-50[73]	26,560	53 Conv	0.75

* reported in mean average precision, † reported in mean accuracy

Using these above equations we develop a novel testing tool, DeepInspect, to inspect a DNN implementing image classification tasks and look for potential confusion and bias errors. We implemented DeepInspect in the Pytorch deep learning framework and Python 2.7. All our experiments were run on Ubuntu 18.04.2 with two TITAN Xp GPUs. For all of our experiments, we set the activation threshold Th to be 0.5 for all datasets and models. We discuss why we choose 0.5 as neuron activation threshold and how different thresholds affect our performance in the section 7.

4 EXPERIMENTAL DESIGN

4.1 Study Subjects

We apply DeepInspect for both multi-label and single-label DNN-based classifications. Under different settings, DeepInspect automatically inspects 8 DNN models for 6 datasets. Table 2 summarizes our study subjects. All the models we used are standard, widely-used models for each dataset. We used pre-trained models as shown in the Table for all settings except for COCO with gender. For COCO with gender model, we used the gender labels from [92] and trained the model in the same way as [92]. imSitu model is a pre-trained ResNet-34 model [85]. There are in total 11,538 entities and 1,788 roles in the imSitu dataset. When inspecting a model trained using imSitu, we only considered the top 100 frequent entities or roles in the test dataset.

Among the 8 DNN models, three are pre-trained relatively more robust models that are trained using adversarial images along with regular images. These models are pre-trained by provably robust training approach proposed by [82]. Three models with different network structures are trained using the CIFAR10 dataset [82].

4.2 Constructing Ground Truth (GT) Errors

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

4.2.1 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 those two classes than between 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:

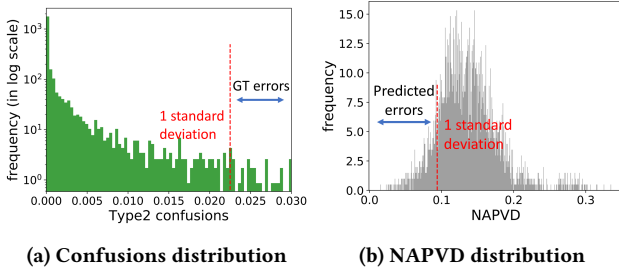


Figure 2: Identifying Type2 confusions for multi-classification applications. LHS shows how we marked the ground truth errors based on Type2 confusion score. RHS shows DeepInspect’s predicted errors based on NAPVD score.

Type1 confusions: In single-label classification, Type1 confusion occurs when an object of class x (e.g., violin) is misclassified to another class y (e.g., cello). For all the objects of class x and y , it can be quantified as: $\text{type1conf}(x, y) = \text{mean}(P(x|y), P(y|x))$ —DNN’s probability to misclassify class y as x 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 vice versa. Note that, this is a bi-directional score, i.e. misclassification of y as x is the same as misclassification of x as y .

Type2 confusions: In multi-label classification, Type2 confusion occurs when an input image contains an object of class x (e.g., mouse) and no object of class y (e.g., keyboard), but the model predicts both classes (see Figure 7). For a pair of classes, this can be quantified as: $\text{type2conf}(x, y) = \text{mean}(P((x, y)|x), P((x, y)|y))$ 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 vice versa. 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. For example, Figure 2a shows type2conf distribution for COCO. The class-pairs with confusion scores greater than 1 standard deviation from the mean-value are marked as pairs truly confused by the model and form our ground truth for confusion errors. For example, in the COCO dataset, there are 80 classes and thus 3160 class pairs ($80 \times 79 / 2$); 178 class-pairs are ground-truth confusion errors.

Note that, unlike 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 a higher chance to be misclassified by the model.

4.2.2 GT of Bias Errors. 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 in error rate between classes x and z and between y and z : $\text{cd}(x, y, z) = |\text{error}(x, z) - \text{error}(y, z)|$, 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 x, y (e.g., man, woman) w.r.t. a third class z (e.g., surfboard).

We also define an aggregated measure **average confusion disparity** (avg_cd) between two classes x and y by summing up the bias between them over all third classes and taking the average:

$$\text{avg_cd}(x, y) := \frac{1}{|O| - 2} \sum_{z \in O, z \neq x, y} \text{cd}(x, y, z).$$

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 the true classification error rate reported 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 one standard deviation from the mean value. Such truly biased pairs form our ground truth for bias errors.

4.3 Evaluating DeepInspect

We evaluate DeepInspect using a set of test images.

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 2b. Class pairs having NAPVD scores lower than 1 standard deviation from the mean score are marked as potential confusion errors.

As discussed in Section 3.3, 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 $\text{avg_bias}(a, b)$ 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) from the mean value of all *Deltas* across all the class-pairs.

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). Similarly to how static analysis warnings are ranked based on their priority levels [63], 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 the ranking we use a cost-effectiveness measure [2], AUCEC (Area Under the Cost-Effectiveness Curve), which has become standard to evaluate rank-based bug-prediction systems [27, 63–66].

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 6 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.

Baseline. We compare DeepInspect *w.r.t.* two baselines:

(i) **MODE-inspired:** A popular way to inspect each image is to inspect a feature vector, which is an output of an intermediate layer [43, 90]. However, abstracting a feature vector per image to the class level is non-trivial. Instead, for a given layer, one could inspect the weight vector ($w_l = [w_l^0, w_l^1, \dots, w_l^n]$) of a class, say l , where the superscripts represent a feature. Similar weight-vectors are used in MODE [43] to compare the difference in feature importance between two image groups. In particular, from the last linear layer before the output layer we extract such per-class weight vectors and compute the pairwise distances between the weight vectors. Using these pairwise distances we calculate confusion and bias metrics as described in Section 3.

(ii) **Random:** We also build a random model that picks random class-pairs for inspection [81] as a baseline.

For AUCEC evaluation, we further 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 lower upper bound of the ranking scheme.

Research Questions. With this experimental setting, we investigate the following three research questions to evaluate DeepInspect for DNN image classifiers:

- **RQ1.** Can DeepInspect distinguish between different classes?
- **RQ2.** Can DeepInspect identify the confusion errors?
- **RQ3.** Can DeepInspect identify the bias errors?

5 RESULTS

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

RQ1. Can DeepInspect distinguish between different classes?

Motivation. The heart of DeepInspect’s error detection technique lies in the fact that the underlying Neuron Activation Probability metric (ρ) captures each class abstraction reasonably well and thus distinguishes between classes that do not suffer from class-level violations. In this RQ we check whether this is indeed true. We also check whether a new metric ρ is necessary, *i.e.*, whether existing neuron-coverage metrics could capture such class separations.

Approach. We evaluate this RQ *w.r.t.* the training data since the DNN behaviors are not tainted with inaccuracies associated with the test images. Thus, all the class-pairs are benign. We evaluate this RQ in three settings: (i) using DeepInspect’s metrics, (ii) neuron-coverage proposed by Pei *et al.* [60], and (iii) other neuron-activation related metrics proposed by DeepGauge [42].

Setting-1. DeepInspect. Our metric, Neuron Activation Probability Matrix (ρ), by construction is designed per class. Hence it would be unfair to directly measure its capability to distinguish between different classes. Thus, we pose this question in slightly a different way, as described below. For multi-label classification,

each image contains multiple class-labels. For example, an image might 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: $\text{coincidence}(L_a, L_b) = \text{mean}(P(L_a, L_b|L_a), P(L_a, L_b|L_b))$.

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 least coincidence in either direction. A low value of coincidence score indicates two class-labels are easy to separate and vice 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 scores.

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 apart 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 [92].

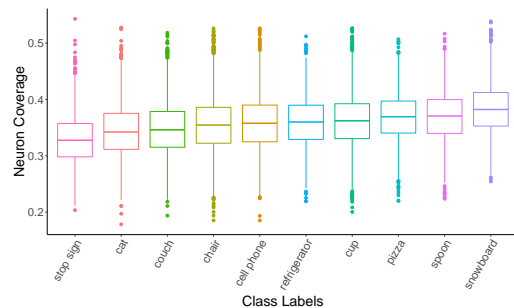


Figure 3: Distribution of neuron coverage per class label, for 10 randomly picked class labels, from the COCO dataset.

Next, we investigate whether popular white-box metrics can distinguish between different classes.

Setting-2. Neuron Coverage (NC) [60] computes the ratio of the union of 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.* to the total neurons. The

activation threshold we use is 0.5. We perform this experiment on COCO and CIFAR-100 to study multi- and single-label classifications. Figure 3 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 [33], 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%

Setting-3. DeepGauge [42]. Ma *et al.* [42] 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 training time; outputs outside this region ($(-\infty, low) \cup (high, +\infty)$) are marked as corner cases. They therefore 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 regions are 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.

We investigate whether each of these metrics can distinguish between different classes by measuring the above metrics for individual input classes following Ma *et al.*’s 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 coverage using test images containing that class; for k-multisection coverage we chose $k = 100$ to scale up the analysis. It should be noted that we also tried $k = 1000$ (which is used in the original DeepGauge paper) and observed similar results (not shown here).

For layer-level coverage, we directly used the input images containing each class, where we select $k = 1$.

Figure 4 shows the results as a histogram of the above five coverage criteria for the COCO dataset. For all 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. Similarly, there are 40 labels with 0 neuron boundary coverage. Therefore, none of the five coverage criteria are an effective way to distinguish between

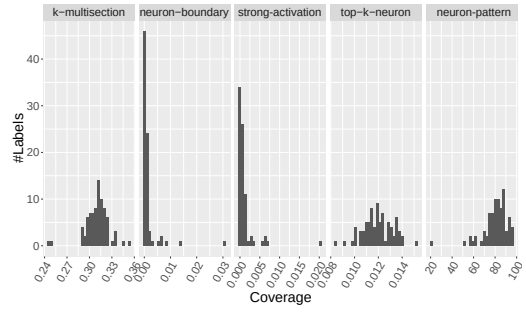


Figure 4: Histogram of DeepGauge [42] multi-granular coverage per class label for COCO dataset

different equivalence classes. The same conclusion was drawn for the CIFAR-100 dataset.

Result 1: *DeepInspect can disambiguate classes better than previous coverage-based metrics for the image classification task.*

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

RQ2. Can DeepInspect identify the confusion errors?

Motivation. To evaluate how well DeepInspect can detect class-level violations, in this RQ, we report DeepInspect’s ability to detect the first type of violation, i.e., Type1/Type2 confusions *w.r.t.* to ground truth confusion errors, as described in Section 4.2.1.

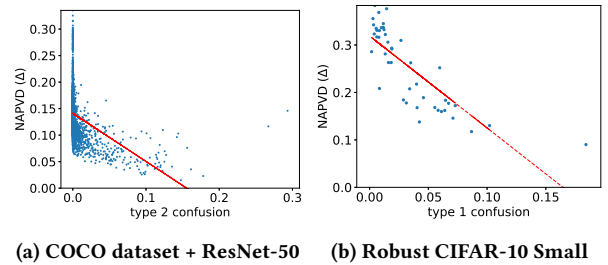


Figure 5: Strong negative Spearman correlation (-0.55 and -0.86) between NAPVD and ground truth confusion scores.

We first explore the correlation between NAPVD and ground truth Type1/Type2 confusion score. Strong correlation has been found for all 8 experimental settings. Figure 5 gives examples on COCO and CIFAR-10. These results indicate that NAPVD can be used to detect confusion errors—lower NAPVD means more confusion.

Approach. By default, DeepInspect reports all the class-pairs with NAPVD scores one standard deviation less than the mean NAPVD score as error-prone (See Figure 2b). In this setting, as the result shown on Table 3, 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 3: 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

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 3, 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 6, 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%.

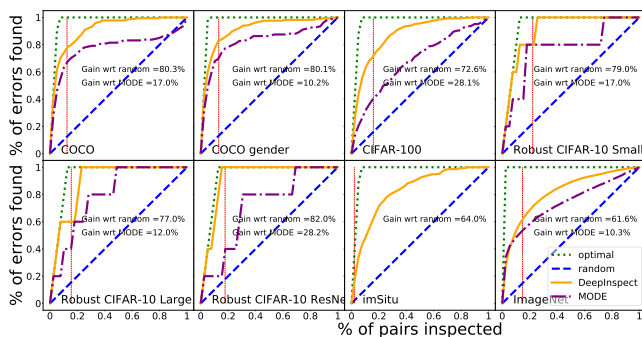


Figure 6: AUCEC plot of Type1/Type2 Confusion errors in three different settings. The red vertical line marks 1-standard deviation less from mean NAPVD score. DeepInspect marks all class-pairs with NAPVD scores less than the red mark as potential errors.

Figure 7 and Figure 8 give some specific confusion errors found by DeepInspect in the COCO and the ImageNet settings. In particular, as shown in Figure 7a, when there is only a keyboard but



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

Figure 7: Confusion errors identified in COCO model. In each pair the second object is mistakenly identified by the model.

no mouse in the image, the COCO model reports both. Similarly, Figure 8a shows confusion errors on (cello, violin). There are several cellos in this image, but the model predicts it to show a violin.



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

Figure 8: Confusion errors identified in the ImageNet model. For each pair, the second object is mistakenly identified by the model.

Across all three relatively more robust CIFAR-10 models DeepInspect identifies (cat, dog), (bird, deer) and (automobile, truck) as buggy 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 on this dataset including the 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.

We also note that the performance of all methods degrades quite a bit on ImageNet. ImageNet is known to have a complex structure, and all the tasks, including image classification and robust image classification [83] usually have inferior performance compared with simpler datasets like CIFAR-10 or CIFAR-100. Due to such inherent complexity, the class representation in the embedded space is less accurate, and thus the relative distance between two classes may not correctly reflect a model's confusion level between two classes.

Result 2: 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.

RQ3. Can DeepInspect identify the bias errors?

Motivation. To assess DeepInspect's ability to detect class-level violations, in this RQ, we report DeepInspect's performance in detecting the second type of violation, *i.e.*, Bias errors as described in Section 4.2.2.

Approach. We evaluate this RQ by estimating a model's bias (avg_bias) using Equation (4) *w.r.t.* the ground truth (avg_cd), computed as

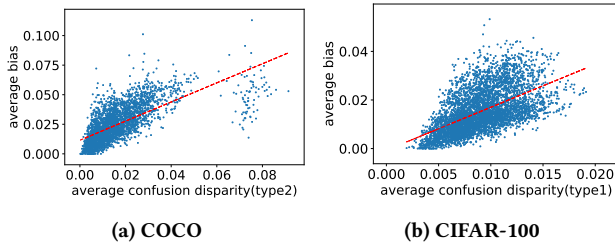


Figure 9: Strong positive Spearman’s correlation (0.76 and 0.62) exist between avg_cd and avg_bias while detecting classification bias.

in Section 4.2.2. We first explore the correlation between pairwise avg_cd and our proposed pairwise avg_bias; Figure 9 shows the results for COCO and CIFAR-10. Similar trends were found in the other datasets we studied. 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 confusion errors.

Table 4: 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	0.034
	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

As in RQ2, we also do a precision-recall analysis *w.r.t.* finding the bias errors across all the datasets. We analyze the precision and recall of DeepInspect when reporting bias errors at the cutoff Top1%(avg_bias) and mean(avg_bias)+standard deviation(avg_bias), respectively. The results are shown in Table 4. 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: DeepInspect detects ground truth suspicious pairs with recall at 75.9% and 71.8% for COCO and imSitu. DeepInspect can report 657(=249+408) total true bias bugs across the two models. DeepInspect outperforms the random baseline by a large margin at both cutoffs. As in the case of detecting confusion errors, there is a significant trade-off between precision

and recall. This can be customized based on user needs. The cost-effectiveness analysis in Figure 10 shows the entire spectrum.

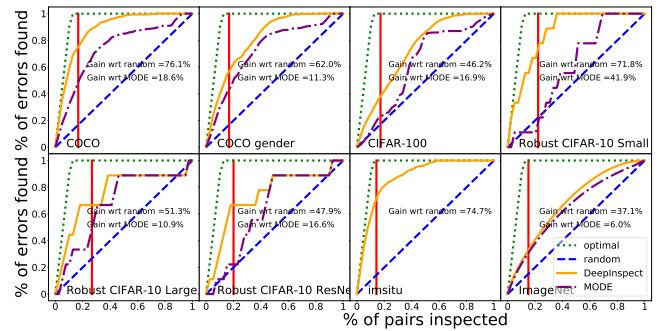


Figure 10: Bias errors detected *w.r.t.* the ground truth of avg_cd beyond one standard deviation from mean.

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.

Inspired by [92], 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 results on imSitu, where we found that the model tends to associate the class “inside” with women while associating the class “outside” with men.



Figure 11: The model classifies the women in these pictures as men in the 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 biases is between Eskimo_dog and rapeseed *w.r.t.* Siberian_husky. The model tends to confuse the two dogs but not Eskimo_dog and rapeseed. This makes sense since Eskimo_dog and Siberian_husky are both dogs so more easily misclassified by the model.

One of the fairness violations of a DNN system can be drastic differences in 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



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.

between two classes 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 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%.*

6 RELATED WORK

Software Testing & Verification of DNNs. Prior research proposed different white-box testing criteria based on neuron coverage [42, 60, 77] and neuron-pair coverage [74]. Sun *et al.* [75] 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[53], which is 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 [25, 28, 61, 80] against adversarial attacks. However, most of the verification efforts are limited to small DNNs and pixel-level properties. It is not obvious how to directly apply these techniques to detect class-level violations.

Adversarial Deep Learning. DNNs are known to be vulnerable to well-crafted inputs called adversarial examples, where the discrepancies are imperceptible to a human but can easily make DNNs fail [14, 17, 24, 31, 34, 40, 51, 52, 54, 57, 58, 62, 76, 86]. Much work has been done to defend against adversarial attacks [4, 10, 15, 18, 20, 23, 45, 47, 55, 59, 72, 84, 93]. Our methods have potential to identify adversarial inputs. Moreover, adversarial examples are usually out of distribution data and not realistic, while we can find both out-distribution and in-distribution corner cases. Further, we can identify a general weakness or bug rather than focusing on crafted attacks that often require a strong attacker model (*e.g.*, the attacker adds noise to a stop sign image).

Interpreting DNNs. There has been much research on model interpretability and visualization [5, 11, 39, 49, 71, 91]. A comprehensive study is presented by Lipton [39]. Dong *et al.* [11] 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.

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. Closest to ours is by Papernot *et al.* [56], who used nearest training points to explain adversarial attacks. In comparison, we analyze the DNN’s dependencies on the entire training/testing data and represent it in Neuron Activation Probability Matrix. We can explain the DNN’s bias and weaknesses by inspecting this matrix. **Evaluating Models’ Bias/Fairness.** Evaluating the bias and fairness of a system is important both from a theoretical and a practical perspective [7, 41, 88, 89]. Related studies first define a fairness criteria and then try to optimize the original objective while satisfying the fairness criteria [3, 12, 13, 21, 36, 46]. These properties are defined either at individual [13, 30, 35] or group levels [9, 21, 87]. In this work, we propose a definition of a bias error for image classification closely related to fairness notions at group-level. Class membership can be regarded as the sensitive feature and the equality that we want to achieve is for the confusion levels of two groups w.r.t. any third group. We showed the potential of DeepInspect to detect such violations.

Galhotra *et al.* [16] 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 work, we use 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.* [56] shows such explainability behind model bias in a single classification setting.

7 DISCUSSION & THREATS TO VALIDITY

Discussion. In the literature, bug detection, debugging, and repair are usually three distinct tasks, and there is a large body of work investigating each separately. In this work, we focus on bug detection for image classifier software. A natural follow-up of our work will be debugging and repair leveraging DeepInspect’s bug detection. We present some preliminary results and thoughts.

A commonly used approach to improving (*i.e.* fixing) image classifiers is active learning, which consists of adding more labeled data by smartly choosing what to label next. In our case, we can use NAPVD to identify the most confusing class pairs, and then target those pairs by collecting additional examples that contain individual objects from the confusing pairs. We download 105 sample images from Google Images that contain isolated examples of these categories so that the model learns to disambiguate them. We retrain the model from scratch using the original training data

and these additional examples. Using this approach, we have some preliminary results on the COCO dataset. After retraining, we find that the type2conf of the top confused pairs reduces. For example, the type2conf(baseball bat, baseball glove) is reduced from 0.23 to 0.16, and type2conf(refrigerator, oven) is reduced from 0.14 to 0.10. Unlike traditional active learning approaches that encourage labeling additional examples near the current decision boundary of the classifier, our approach encourages the labeling of problematic examples based on confusion bugs.

Another potential direction to explore is to use DeepInspect in tandem with debugging & repair tools for DNN models like MODE [43]. DeepInspect enables the user to focus debugging effort on the vulnerable classes even in the absence of labeled data. For instance, once DeepInspect identifies the vulnerable class-pairs, one can use the GAN-based approach proposed in MODE to generate more training data from these class-pairs, apply MODE to identify the most vulnerable features in these pairs to select for retraining.

We have also explored how the neuron coverage threshold(th) used in computing $NAPVD$ affects our performance in detecting confusion and bias errors. We studied one multi-label classification task COCO and one single-label classification task CIFAR-100. Table 5, 6, 7, 8 show how our precision and recall change when using different neuron coverage thresholds (th). We observed that for CIFAR-100 and COCO that DeepInspect’s accuracies are overall stable at $0.4 \leq th \leq 0.75$. With smaller $th (< 0.25)$, too many neurons are activated pulling the per-class activation-probability-vectors closer to each other. In contrast, with higher $th (> 0.75)$, important activation information gets lost. Thus, we select $th = 0.5$ for all the other experiments to avoid either issue.

Table 5: DeepInspect impact of neuron coverage threshold on detecting confusion errors for COCO

NC threshold	NAPVD < mean-1std				Top 1%			
	TP	FP	Precision	Recall	TP	FP	Precision	Recall
0.25	36	18	0.67	0.20	23	8	0.74	0.13
0.40	150	215	0.41	0.84	31	0	1	0.17
0.50	138	256	0.35	0.78	31	0	1	0.17
0.60	137	264	0.34	0.77	30	1	0.97	0.17
0.75	135	271	0.33	0.76	29	2	0.94	0.16

Table 6: DeepInspect impact of neuron coverage threshold on detecting confusion errors for CIFAR-100

NC threshold	NAPVD < mean-1std				Top 1%			
	TP	FP	Precision	Recall	TP	FP	Precision	Recall
0.25	188	629	0.23	0.66	34	15	0.69	0.12
0.40	197	550	0.26	0.69	39	10	0.80	0.14
0.50	206	584	0.26	0.72	39	10	0.80	0.14
0.60	211	596	0.26	0.74	37	12	0.76	0.13
0.75	195	604	0.24	0.68	37	12	0.76	0.13

Threats to Validity. We only test DeepInspect on 6 datasets under 8 settings. We include both single-class and multi-class as well

Table 7: DeepInspect impact of neuron coverage threshold on detecting bias errors for COCO

NC threshold	avg_bias > mean+1std				Top 1%			
	TP	FP	Precision	Recall	TP	FP	Precision	Recall
0.25	218	280	0.438	0.665	26	6	0.812	0.079
0.40	260	275	0.486	0.793	20	12	0.625	0.061
0.50	249	278	0.472	0.759	24	8	0.75	0.073
0.60	190	273	0.410	0.579	24	8	0.75	0.073
0.75	197	54	0.785	0.601	32	0	1	0.098
0.90	201	102	0.663	0.592	32	0	1	0.094

Table 8: DeepInspect impact of neuron coverage threshold on detecting bias errors for CIFAR-100

NC threshold	avg_bias > mean+1std				Top 1%			
	TP	FP	Precision	Recall	TP	FP	Precision	Recall
0.25	289	569	0.337	0.355	18	32	0.36	0.022
0.40	272	545	0.333	0.334	27	23	0.54	0.033
0.50	310	543	0.363	0.380	29	21	0.58	0.036
0.60	279	473	0.371	0.342	26	24	0.54	0.032
0.75	276	455	0.378	0.339	29	21	0.58	0.036
0.90	179	587	0.234	0.220	12	38	0.24	0.015

as regular and robust models to address these threats as much as possible.

Another limitation is that DeepInspect needs to decide thresholds for both confusion errors and bias errors, and a threshold for discarding low-confusion triplets in the estimation of avg_bias. Instead of choosing fixed threshold, we mitigate this threat by choosing thresholds that are one standard deviation from the corresponding mean values and, also, reporting performance at top1%.

The task of accurately classifying any image is notoriously difficult. We simplify the problem by testing the DNN model only for the classes that it has seen during training. For example, while training, if a DNN does not learn to differentiate between black vs. brown cows (*i.e.*, all the cow images only have label cow and they are treated as belonging to the same class by the DNN), DeepInspect will not be able to test these sub-groups.

8 CONCLUSION

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

ACKNOWLEDGMENTS

This work is supported in part by NSF CNS-1563555, CCF-1815494, CNS-1842456, CCF-1845893, and CCF-1822965. Any opinions, findings, conclusions, or recommendations expressed herein are those of the authors, and do not necessarily reflect those of the US Government or NSF.

REFERENCES

- [1] 2017. Base pretrained models and datasets in pytorch. <https://github.com/aaron-xichen/pytorch-playground>
- [2] Erik Arisholm, Lionel C. Briand, and Eivind B. Johannessen. 2010. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *JSS* 83, 1 (2010), 2–17.
- [3] Solon Barocas, Moritz Hardt, and Arvind Narayanan. 2018. *Fairness and Machine Learning*. fairmlbook.org. <http://www.fairmlbook.org>.
- [4] Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya Nori, and Antonio Criminisi. 2016. Measuring neural net robustness with constraints. In *Advances in Neural Information Processing Systems*. 2613–2621.
- [5] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. 2017. Network Dissection: Quantifying Interpretability of Deep Visual Representations. In *Computer Vision and Pattern Recognition*.
- [6] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35, 8 (2013), 1798–1828.
- [7] Yuriy Brun and Alexandra Meliou. 2018. Software Fairness. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Lake Buena Vista, FL, USA) (ESEC/FSE 2018). ACM, New York, NY, USA, 754–759. <https://doi.org/10.1145/3236024.3264838>
- [8] Joy Buolamwini and Timnit Gebru. 2018. Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification. In *FAT*.
- [9] T. Calders, F. Kamiran, and M. Pechenizkiy. 2009. Building Classifiers with Interdependency Constraints. In *2009 IEEE International Conference on Data Mining Workshops*. 13–18.
- [10] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 39–57.
- [11] Yinpeng Dong, Hang Su, Jun Zhu, and Fan Bao. 2017. Towards interpretable deep neural networks by leveraging adversarial examples. *arXiv preprint arXiv:1708.05493* (2017).
- [12] Michele Donini, Luca Oneto, Shai Ben-David, John Shawe-Taylor, and Massimiliano Pontil. 2018. 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*. 2796–2806. <http://papers.nips.cc/paper/7544-empirical-risk-minimization-under-fairness-constraints>
- [13] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard S. Zemel. 2012. Fairness Through Awareness. In *Proceedings of the Innovations in Theoretical Computer Science Conference* abs/1104.3913 (2012), 214–226.
- [14] Ivan Evtimov, Kevin Eykholt, Earlene Fernandes, Tadayoshi Kohno, Bo Li, Atul Prakash, Amir Rahmati, and Dawn Song. 2017. Robust Physical-World Attacks on Machine Learning Models. *arXiv preprint arXiv:1707.08945* (2017).
- [15] Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. 2017. Detecting Adversarial Samples from Artifacts. *arXiv preprint arXiv:1703.00410* (2017).
- [16] Sainyam Galhotra, Yuriy Brun, and Alexandra Meliou. 2017. Fairness testing: testing software for discrimination. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM, 498–510.
- [17] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations (ICLR)*.
- [18] Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick McDaniel. 2017. On the (statistical) detection of adversarial examples. *arXiv preprint arXiv:1702.06280* (2017).
- [19] Loren Grush. 2015. Google engineer apologizes after Photos app tags two black people as gorillas. (2015). <https://www.theverge.com/2015/7/1/8880363/google-apologizes-photos-app-tags-two-black-people-gorillas>
- [20] Shixiang Gu and Luca Rigazio. 2015. Towards deep neural network architectures robust to adversarial examples. In *International Conference on Learning Representations (ICLR)*.
- [21] Moritz Hardt, Eric Price, and Nathan Srebro. 2016. Equality of Opportunity in Supervised Learning. In *Proceedings of the 30th International Conference on Neural Information Processing Systems (Barcelona, Spain) (NIPS'16)*. USA, 3323–3331.
- [22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
- [23] Warren He, James Wei, Xinyun Chen, Nicholas Carlini, and Dawn Song. 2017. Adversarial Example Defenses: Ensembles of Weak Defenses Are Not Strong. In *Proceedings of the 11th USENIX Conference on Offensive Technologies* (Vancouver, BC, Canada) (WOOT'17). USENIX Association, Berkeley, CA, USA, 15–15. <http://dl.acm.org/citation.cfm?id=3154768.3154783>
- [24] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. 2017. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284* (2017).
- [25] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. 2017. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification*. Springer, 3–29.
- [26] Pooja Kamavisdar, Sonam Saluja, and Sonu Agrawal. 2013. A survey on image classification approaches and techniques. *International Journal of Advanced Research in Computer and Communication Engineering* 2, 1 (2013), 1005–1009.
- [27] Yasutaka Kamei, Shinsuke Matsumoto, Akito Monden, Ken-ichi Matsumoto, Bram Adams, and Ahmed E Hassan. 2010. Revisiting common bug prediction findings using effort-aware models. In *2010 IEEE International Conference on Software Maintenance*. IEEE, 1–10.
- [28] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2017. *Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks*. Springer International Publishing, Cham, 97–117.
- [29] Jinhan Kim, Robert Feldt, and Shin Yoo. 2019. Guiding deep learning system testing using surprise adequacy. In *Proceedings of the 41st International Conference on Software Engineering*. IEEE Press, 1039–1049.
- [30] Michael P. Kim, Omer Reingold, and Guy N. Rothblum. 2018. Fairness Through Computationally-Bounded Awareness. *32nd Conference on Neural Information Processing Systems (NeurIPS 2018)* (2018).
- [31] Jernej Kos, Ian Fischer, and Dawn Song. 2017. Adversarial examples for generative models. *arXiv preprint arXiv:1702.06832* (2017).
- [32] Alex Krizhevsky. 2012. Learning Multiple Layers of Features from Tiny Images. *University of Toronto* (05 2012).
- [33] William H. Kruskal and W. Allen Wallis. 1952. Use of Ranks in One-Criterion Variance Analysis. *J. Amer. Statist. Assoc.* 47, 260 (1952), 583–621. <https://doi.org/10.1080/01621459.1952.10483441> <http://www.tandfonline.com/doi/pdf/10.1080/01621459.1952.10483441>
- [34] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2017. Adversarial examples in the physical world. In *Workshop track at International Conference on Learning Representations (ICLR)*.
- [35] Matt J Kusner, Joshua Loftus, Chris Russell, and Ricardo Silva. 2017. Counterfactual Fairness. In *Advances in Neural Information Processing Systems 30*. 4066–4076.
- [36] Alexandre Louis Lamy, Ziyuan Zhong, Aditya Krishna Menon, and Nakul Verma. 2019. Noise-tolerant fair classification. *CoRR* abs/1901.10837 (2019). <http://arxiv.org/abs/1901.10837>
- [37] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
- [38] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft coco: Common objects in context. In *European conference on computer vision*. Springer, 740–755.
- [39] Zachary C Lipton. 2016. The myths of model interpretability. *Proceedings of the 33rd International Conference on Machine Learning Workshop* (2016).
- [40] Jiajun Lu, Hussein Sibai, Evan Fabry, and David Forsyth. 2017. 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*.
- [41] Binh Thanh Luong, Salvatore Ruggieri, and Franco Turini. 2011. 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, 502–510.
- [42] 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. 2018. DeepGauge: Multi-granularity Testing Criteria for Deep Learning Systems. (2018), 120–131. <https://doi.org/10.1145/3238147.3238202>
- [43] Shiqing Ma, Yingqi Liu, Wen-Chuan Lee, Xiangyu Zhang, and Ananth Grama. 2018. MODE: Automated Neural Network Model Debugging via State Differential Analysis and Input Selection. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Lake Buena Vista, FL, USA) (ESEC/FSE 2018). ACM, New York, NY, USA, 175–186. <https://doi.org/10.1145/3236024.3236082>
- [44] MalletsDarker. 2018. I took a few shots at Lake Louise today and Google offered me this panorama. (2018). https://www.reddit.com/r/funny/comments/7r9ptc/i_took_a_few_shots_at_lake_louise_today_and/dsvv1nw/
- [45] Chengzhi Mao, Ziyuan Zhong, Junfeng Yang, Carl Vondrick, and Baishakhi Ray. 2019. Metric Learning for Adversarial Robustness. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Curran Associates, Inc., 478–489. <http://papers.nips.cc/paper/8339-metric-learning-for-adversarial-robustness.pdf>
- [46] Aditya Krishna Menon and Robert C. Williamson. 2018. The cost of fairness in binary classification. In *Conference on Fairness, Accountability and Transparency, FAT 2018, 23-24 February 2018, New York, NY, USA*. 107–118. <http://proceedings.mlr.press/v81/menon18a.html>
- [47] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. 2017. On detecting adversarial perturbations. In *International Conference on Learning Representations (ICLR)*.
- [48] Thomas M. Mitchell. 1997. *Machine Learning* (1 ed.). McGraw-Hill, Inc., New York, NY, USA.

- [49] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. 2017. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing* (2017).
- [50] Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, 807–814.
- [51] Nina Narodytska and Shiva Prasad Kasiviswanathan. 2016. Simple black-box adversarial perturbations for deep networks. In *Workshop on Adversarial Training, NIPS 2016*.
- [52] Anh Nguyen, Jason Yosinski, and Jeff Clune. 2015. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 427–436.
- [53] Augustus Odena, Catherine Olsson, David Andersen, and Ian Goodfellow. 2019. TensorFuzz: Debugging Neural Networks with Coverage-Guided Fuzzing. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.), Vol. 97. PMLR, Long Beach, California, USA, 4901–4911. <http://proceedings.mlr.press/v97/odena19a.html>
- [54] Nicolas Papernot, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Fartash Faghri, Alexander Matyasko, Karen Hambardzumyan, Yi-Lin Juang, Alexey Kurakin, Ryan Sheatsley, et al. 2016. cleverhans v2. 0.0: an adversarial machine learning library. *arXiv preprint arXiv:1610.00768* (2016).
- [55] Nicolas Papernot and Patrick McDaniel. 2017. Extending Defensive Distillation. *arXiv preprint arXiv:1705.05264* (2017).
- [56] Nicolas Papernot and Patrick McDaniel. 2018. Deep k-Nearest Neighbors: Towards Confident, Interpretable and Robust Deep Learning. *arXiv preprint arXiv:1803.04765* (2018).
- [57] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. 2017. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ACM, 506–519.
- [58] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 372–387.
- [59] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2016. Distillation as a defense to adversarial perturbations against deep neural networks. In *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 582–597.
- [60] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. (2017), 1–18. <https://doi.org/10.1145/3132747.3132785>
- [61] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Towards Practical Verification of Machine Learning: The Case of Computer Vision Systems. *arXiv preprint arXiv:1712.01785* (2017).
- [62] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. 2018. Certified defenses against adversarial examples. *6th International Conference on Learning Representations (ICLR)* (2018).
- [63] Foyzur Rahman and Premkumar Devanbu. 2013. How, and why, process metrics are better. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 432–441.
- [64] Foyzur Rahman, Daryl Posnett, Israel Herraiz, and Premkumar Devanbu. 2013. Sample size vs. bias in defect prediction. In *Proceedings of the 2013 9th joint meeting on foundations of software engineering*. ACM, 147–157.
- [65] F. Rahman, D. Posnett, A. Hindle, E. Barr, and P. Devanbu. 2011. 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, 322–331.
- [66] Baishakhi Ray, Vincent Hellendoorn, Saheel Godhane, Zhaopeng Tu, Alberto Bacchelli, and Premkumar Devanbu. 2016. On the “naturalness” of buggy code. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 428–439.
- [67] Adam Rose. 2010. Are Face-Detection Cameras Racist? (2010). <http://content.time.com/time/business/article/0,8599,1954643,00.html>
- [68] Amir Rosenfeld, Richard S. Zemel, and John K. Tsotsos. 2018. The Elephant in the Room. *CoRR abs/1808.03305* (2018). [arXiv:1808.03305](http://arxiv.org/abs/1808.03305) <http://arxiv.org/abs/1808.03305>
- [69] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1988. Learning representations by back-propagating errors. *Cognitive modeling* 5, 3 (1988), 1.
- [70] 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. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252. <https://doi.org/10.1007/s11263-015-0816-y>
- [71] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. 2017. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. In *2017 IEEE International Conference on Computer Vision (ICCV)*, 618–626. <https://doi.org/10.1109/ICCV.2017.74>
- [72] Uri Shaham, Yutaro Yamada, and Sahand Negahban. 2015. Understanding adversarial training: Increasing local stability of neural nets through robust optimization. *arXiv preprint arXiv:1511.05432* (2015).
- [73] Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*.
- [74] Youcheng Sun, Xiaowei Huang, and Daniel Kroening. 2018. Testing Deep Neural Networks. *arXiv preprint arXiv:1803.04792* (2018).
- [75] Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. 2018. Concolic Testing for Deep Neural Networks. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (Montpellier, France) (ASE 2018)*. ACM, New York, NY, USA, 109–119. <https://doi.org/10.1145/3238147.3238172>
- [76] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. 2014. Intriguing properties of neural networks. In *International Conference on Learning Representations (ICLR)*.
- [77] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. DeepTest: Automated testing of deep-neural-network-driven autonomous cars. In *International Conference of Software Engineering (ICSE), 2018 IEEE conference on*. IEEE.
- [78] Grigoris Tsoumakas and Ioannis Katakis. 2007. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDM)* 3, 3 (2007), 1–13.
- [79] Jingyi Wang, Guoliang Dong, Jun Sun, Xinyu Wang, and Peixin Zhang. 2019. Adversarial sample detection for deep neural network through model mutation testing. In *Proceedings of the 41st International Conference on Software Engineering*. IEEE Press, 1245–1256.
- [80] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Formal Security Analysis of Neural Networks using Symbolic Intervals. (2018).
- [81] Ian H Witten and Eibe Frank. 2005. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.
- [82] Eric Wong, Frank Schmidt, Jan Hendrik Metzen, and J. Zico Kolter. 2018. 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., 8410–8419. <http://papers.nips.cc/paper/8060-scaling-provable-adversarial-defenses.pdf>
- [83] Cihang Xie, Yuxin Wu, Laurens van der Maaten, Alan L. Yuille, and Kaiming He. 2019. Feature Denoising for Improving Adversarial Robustness. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- [84] Weilin Xu, David Evans, and Yanjun Qi. 2017. Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks. *arXiv preprint arXiv:1704.01155* (2017).
- [85] Mark Yatskar, Luke Zettlemoyer, and Ali Farhadi. 2016. Situation Recognition: Visual Semantic Role Labeling for Image Understanding. In *Conference on Computer Vision and Pattern Recognition*.
- [86] X. Yuan, P. He, Q. Zhu, and X. Li. 2019. Adversarial Examples: Attacks and Defenses for Deep Learning. *IEEE Transactions on Neural Networks and Learning Systems* (2019), 1–20. <https://doi.org/10.1109/TNNLS.2018.2886017>
- [87] Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez Rodriguez, and Krishna P. Gummadi. 2017. Fairness Beyond Disparate Treatment & Disparate Impact: Learning Classification Without Disparate Mistreatment. In *Proceedings of the 26th International Conference on World Wide Web (Perth, Australia)*, 1171–1180.
- [88] Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez Rodriguez, and Krishna P Gummadi. 2017. Fairness constraints: Mechanisms for fair classification. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics ((AISTATS) 2017)*, Vol. 54. JMLR.
- [89] Rich Zemel, Yu Wu, Kevin Swersky, Toni Pitassi, and Cynthia Dwork. 2013. Learning Fair Representations. In *Proceedings of the 30th International Conference on Machine Learning*. 325–333.
- [90] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. 2018. DeepRoad: GAN-based Metamorphic Autonomous Driving System Testing. *arXiv preprint arXiv:1802.02295* (2018).
- [91] Quan-shi Zhang and Song-Chun Zhu. 2018. Visual interpretability for deep learning: a survey. *Frontiers of Information Technology & Electronic Engineering* 19, 1 (2018), 27–39.
- [92] Jieyu Zhao, Tianlu Wang, Mark Yatskar, Vicente Ordonez, and Kai-Wei Chang. 2017. 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*, 2941–2951. <https://www.aclweb.org/anthology/D17-1319>
- [93] Stephan Zheng, Yang Song, Thomas Leung, and Ian Goodfellow. 2016. Improving the robustness of deep neural networks via stability training. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4480–4488.