Understanding Spatial Robustness of Deep Neural Networks

Ziyuan Zhong, Yuchi Tian, Baishakhi Ray

Abstract—Deep Neural Networks (DNNs) are being deployed in a wide range of settings today, from safety-critical applications like autonomous driving to commercial applications involving image classifications. However, recent research has shown that DNNs can be brittle to even slight variations of the input data. Therefore, rigorous testing of DNNs has gained widespread attention. While DNN robustness under norm-bound perturbation got significant attention over the past few years, our knowledge is still limited when natural variants of the input images come. These natural variants, e.g., a rotated or a rainy version of the original input, are especially concerning as they can occur naturally in the field without any active adversary and may lead to undesirable consequences. Thus, it is important to identify the inputs whose small variations may lead to erroneous DNN behaviors. The very few studies that looked at DNN’s robustness under natural variants, however, focus on estimating the overall robustness of DNNs across all the test data rather than localizing such error-producing points. This work aims to bridge this gap.

To this end, we study the local per-input robustness properties of the DNNs and leverage those properties to build a white-box (DEEPROBUST-W) and a black-box (DEEPROBUST-B) tool to automatically identify the non-robust points. Our evaluation of these methods on nine DNN models spanning three widely used image classification datasets shows that they are effective in flagging points of poor robustness. In particular, DEEPROBUST-W and DEEPROBUST-B are able to achieve an F1 score of up to 91.4% and 99.1%, respectively. We further show that DEEPROBUST-W can be applied to a regression problem in a domain beyond image classification. Our evaluation on three self-driving car models demonstrates that DEEPROBUST-W is effective in identifying points of poor robustness with F1 score up to 78.9%.

Index Terms—Deep Neural Networks, Software Testing, Robustness of DNNs.

1 INTRODUCTION

Deep Neural Networks (DNNs) have achieved an unprecedented level of performance over the last decade in many sophisticated areas such as image recognition [1], self-driving cars [2] and playing complex games [3]. These advances have also motivated companies to adapt their software development flows to incorporate AI components [4]. This trend has, in turn, spawned a new area of research within software engineering addressing the quality assurance of DNN components [5]–[16].

Notwithstanding the impressive capabilities of DNNs, recent research has shown that DNNs can be easily fooled, i.e., made to mispredict, with a little variation of the input data [5], [17], [18]—either adding a norm-bound pixel-level perturbation into the original input [17], [19], [20], or with natural variants of the inputs, e.g., rotating an image, changing the lighting conditions, etc. [5], [18]. The natural variants are especially concerning as they can occur naturally in the field without any active adversary and may lead to serious consequences [6], [7].

While norm-bound perturbation based DNN robustness is relatively well-studied, our knowledge of DNN robustness under the natural variations is still limited—we do not know which images are more robust than others, what their characteristics are, etc. For example, consider Figure 1 although the original bird image (a) is predicted correctly by a DNN, its different rotated variations in images (b)-(d) are mispredicted to three different classes. This makes the original image (a) very weak as far as robustness is concerned. In contrast, the bird image (e) and all its rotated versions (generated by the same degrees of rotation) in Figure 1(f)-(h) are correctly classified. Thus, the original image (e) is quite robust. It is important to distinguish between such robust vs. non-robust images, as the non-robust ones can induce errors with the slightest natural variations.

Existing literature, however, focuses on estimating the
overall robustness of DNNs across all the test data \cite{18, 21, 22}. From a traditional software point of view, this is analogous to estimating how buggy a software is without actually localizing the bugs. Our current work tries to bridge this gap by localizing the non-robust points in the input space that pose significant threats to a DNN model’s robustness. However, unlike traditional software where bug localization is performed in program space, we identify the non-robust inputs in the data space. As a DNN is a combination of data and architecture, and the architecture is largely uninterpretable, we restrict our study of non-robustness to the input space. To this end, we first quantify the local (per input) robustness property of a DNN. First, we treat all the natural variants of an input image as its neighbors. Then, for each input data, we consider a population of its neighbors and measure the fraction of this population classified correctly by the DNN - a high fraction of correct classifications indicates good robustness (Figure 1e) and vice versa (Figure 1a). We term this measure neighbor accuracy. Using this metric, we study different local robustness properties of the DNNs and analyze how the weak, a.k.a. non-robust, points differ characteristically from their robust counterparts. Given that the number of natural neighbors of an image can be potentially infinite, first we performed a more controlled analysis by keeping the natural variants limited to spatially transformed images generated by rotation and translation, following the previous work \cite{18, 21, 22}. Such controlled experiments help us to explore different robustness properties while systematically varying transformation parameters.

Our analysis with three well-known object recognition datasets across three popular DNN models, i.e., a total of nine DNN-dataset combinations, reveal several interesting properties of local robustness of a DNN w.r.t. natural variants:

- Local robustness properties seem to be intrinsic to the dataset rather than the model, i.e., most of the robust points remain robust across different models and vice versa.
- The neighbors of a weaker point are not necessarily classified to one single incorrect class. In fact, the weaker the point is its neighbors (mis)classifications become more diverse.
- The weak points are concentrated towards the class decision boundaries of the DNN in the feature space.

Based on these findings, we further develop two techniques (a black-box and a white-box) that can localize the points of poor robustness, thereby providing a means of input-specific, real-time feedback about robustness to the end-user. Our white-box and black-box detectors can identify weak, a.k.a. non-robust, points with f1 score up to 91.4% and 99.1%, respectively, at neighbor accuracy cutoff 0.75. To further check the generalizability of our technique, we aim to detect weak points w.r.t. a self-driving car application where we generated natural input variants by adding rain and fog. Note that these are more complex image transformations, and also the model works in a regression setting instead of classification. These models take an image as input, and output a driving angle. Our white-box detector can identify weak points with f1 score up to 78.9%.

In summary, we make the following contributions:

- We conduct an empirical study to understand the local robustness properties of DNNs under natural variations.
- We develop a white-box (DEEPROBUST-W) and a black-box (DEEPROBUST-B) method to automatically detect weak points.
- We present a detailed evaluation of our methods on three DNN models across three image classification datasets. To check the generalizability of our findings, we further evaluate DEEPROBUST-W in a more complex experimental setting with non-spatial transformations (i.e., rain and fog), different modeling objectives (i.e., regression), and safety-critical application (i.e., self-driving car). We find that DEEPROBUST can successfully detect weak points with reasonably good precision and recall.
- We made our analysis script and code public at https://github.com/deeprobust/DeepRobust.

2 BACKGROUND: DNN TESTING

Many existing studies have proposed different techniques to generate test data inputs by perturbing existing input images for a DNN and used them to evaluate the robustness of the DNN. Depending on how the input image is perturbed, the techniques for generating DNN test data can be classified into three broad categories:

- **i)** Adversarial inputs are typically generated by norm-based perturbation techniques \cite{17, 20, 23–25}, where some pixels of an input image (I) are perturbed by norm-based distance (l1, l2 or linf) such that the distance between the perturbed image and I is ≤ ϵ, where ϵ is a small positive value. These adversarial examples are used to expose the security vulnerabilities of DNNs.

- **ii)** Natural variations are generated through a variety of image transformations, and are used to evaluate the robustness of DNNs under such variations \cite{6, 18, 27}. Sources of these variations include changes in camera configuration, or variations in background or ambient conditions. The transformations simulating these variations could be spatial, such as rotation, translations, mirroring, shear, and scaling on images, or non-spatial transformations, such as changes in the brightness or contrast of an image. Here we first focus on spatial transformations as opposed to adversarial one for two reasons. First, compared with adversarial examples, which is fairly contrived, spatial transformations are more likely to arise in more benign environments. Second, using simple parametric spatial transformations like rotations and translations, it is easier to systematically explore the local robustness properties. Later, to emulate a more natural variation we add fog and rain on the images of self-driving car dataset and evaluate our method’s generalizability.

- **iii)** GAN-based image generation techniques use Generative Adversarial Network (GAN) to synthesize images. GAN is one class of generative models trained as a minimax two-player game between a generative model and a discriminative model \cite{28}. GAN-based image generation has been successfully used to generate DNN test data instances \cite{7}.

**Standard Accuracy vs. Robust Accuracy.** Standard accuracy measures how accurately an ML model predicts the correct classes of the instances in a given test dataset. Robust, a.k.a. adversarial accuracy, estimates how accurately an ML...
model classifies the generated variants [30]. In this paper, we adopt a point-wise robust accuracy measure, neighbor accuracy, to quantify the robustness of a DNN for the neighbors around each data point.

3 METHODOLOGY

3.1 Terminology

Original Data Point: An original data point represents an original un-modified data instance (image in our case) in the studied dataset. The original data points can come from training, validation, or the testing dataset, depending on the experimental setting. In Figure 2, the triangle in the center is an original data point.

Neighbors: Neighbors are images generated by the natural variations, e.g., spatial transformations applied to an original image. Since the transformation parameters are continuous (e.g., degree of rotations), there can be an infinite number of neighbors per image. In Figure 2, the small circles around an original data point represent its neighbors.

Neighbor Accuracy: We define neighbor accuracy as the percentage of its neighbors, including itself, that can be correctly classified by the DNN model. Figure 2 illustrates this; here, red small circles indicate misclassified neighbors, while the green small circles are correctly classified ones. The figure shows that there are only five neighbors per original data point. In the left-hand-side diagram, four out of five neighbors are correctly classified by the given DNN model. If the original data point is correctly classified as well, the neighbor accuracy of the original data is (5/6=) 83.3%. Similarly, in Figure 2 (right), four out of the five neighbors have been misclassified by the model; if the original data point is misclassified, the neighbor accuracy is (1/6=) 16.6%.

![Fig. 2: Illustrating our terminologies. The triangles are original points, and the small circles are their neighbors generated by natural variations. The light-green region is robust with higher neighbor accuracy, while the light-red region is vulnerable. The corresponding original points are robust and non-robust accordingly.](image)

Robustness. An original data point is strong, a.k.a. robust, w.r.t. the DNN model under test if its neighbor accuracy is higher than a pre-defined threshold. Conversely, a weak, a.k.a. non-robust, point has the neighbor accuracy lower than a pre-defined threshold. For example, at 0.75 neighbor accuracy threshold, the black triangle in Figure 2 is a strong point, and the grey triangle is a weak point.

A region contains an original point and all of its neighbors. If the original point is strong (weak), we call the corresponding region as a robust (weak) region. In Figure 2, the light green region is robust while the light red region is weak.

Neighbor Diversity: For multi-class classification task, different neighbors of an original point can be misclassified to different classes. Neighbor Diversity score measures how many diverse classes a point’s neighbors are classified, and is formally computed using Simpson Diversity Index (\(\lambda\)) [31]:

\[
\lambda = \sum_{i=1}^{k} \frac{p_i^2}{i}
\]

where \(k\) is the total number of possible classes and \(p_i\) is the probability of an image’s neighbors being predicted to be class \(i\). Large Simpson Index means low diversity. Let’s consider we have three possible classes A, B, and C. Assume an image has 4 neighbors. Including the original image, there are 5 images in total. If two of the five images are classified as A, and rest are classified as B, then \(\lambda = (2/5)^2 + (3/5)^2 + (0/5)^2 = 0.52\). In contrast, if two of them are classified as A, and two are classified as B, and one is classified as C then \(\lambda = (2/5)^2 + (2/5)^2 + (1/5)^2 = 0.36\). Clearly, the latter case is more diverse and thus, has a lower \(\lambda\) score.

Feature Representation: In a DNN, the neurons’ output in each layer capture different abstract representation of the raw input, which are commonly known as features, extracted by the current layer and all the preceding layers. Each layer’s output forms the corresponding feature space. For a given input data point, we consider the output of the DNN’s second-to-last layer as its feature representation or feature vector.

3.2 Data Collection

Neighbor Generation: For the image classification tasks, for each original image point, we generate its neighbors by combining two types of spatial transformations: rotation and translation. We carefully choose these two types as representatives of non-linear and linear spatial transformations, respectively, following Engstrom et al. [18]. In particular, following them, we generate a neighbor by randomly rotating the original point by \(t\) (in \([-30, 30]\)) degrees, shifting it by \(dx\) (about 10% of the original image’s width i.e. \([-3, 3]\)) pixels horizontally, and shifting it by \(dy\) (about 10% of the original image’s height i.e. \([-3, 3]\)) pixels vertically. It should be noted that for image classification it is standard in the literatures [18], [32], [33] to assume that the transformed image has the same label as the original one. As the transformation parameters are continuous, there can be infinite neighbors of an original data point. Hence, we sample \(m\) neighbors for each original data point. We explore the impact of \(m\) in RQ3.

For the self-driving-car task where the model predicts steering angle, for each original image point, we generate 50% neighbors with rain effect and the rest 50% with fog effects. We adopt a widely used self-driving car data augmentation package, Automold [34], for adding these effects where we randomly vary the degrees of the added effect. For the rain effect, we set “rain_type=heavy” and everything else as default. For the fog effect, we set everything as default.

Estimating Neighbor Accuracy: To compute the neighbor accuracy of a data point for a given DNN model, we first generate its neighbor samples by applying different transformations—spatial for image classification and rain
or fog for self-driving-car application. Then we feed these generated neighbors into the DNN model and compute the accuracy by comparing the DNN’s output with the label of the original data point. For self-driving-car application, we follow the technique described in DeepTest [6]. More specifically, if the predicted steering angle of the transformed image is within a threshold to the original image, we consider it as correct. This ensures that any small variations of steering angle are tolerated in the predicted results. We then compute neighbour accuracy = #correct predictions / original point + total neighbours.

3.3 Classifying Robust vs. Weak Points

We propose two methods, DEEPROBUST-W and DEEPROBUST-B, to identify whether an unlabeled input is strong or weak w.r.t. a DNN in real time. If a test image is identified as a weak point, although it may be classified correctly by the pre-trained model, this image is in a vulnerable region where a slight change to this image may cause the pre-trained DNN to misclassify the changed input.

3.3.1 DEEPROBUST-W: White-box Classifier

This is a binary classifier designed to classify an image (in particular, image feature vector) as a strong or weak point. Here, we assume that we have white box access to the DNN under test to extract the feature vectors of the input images from the DNN. These feature vectors are given as inputs to DEEPROBUST-W. Figure 3 shows the workflow.

**Training:** During training of DEEPROBUST-W, we first feed all the original training images and their neighbors to the DNN under test. From the DNN outputs, we compute the neighbor accuracy for each data point in the training set and label each point strong/weak depending on whether its neighbor accuracy is higher/lower than a predefined threshold. For each original data point, we also extract the output of the DNN’s second-to-last layer as its feature vector. We use these vectors as inputs to train DEEPROBUST-W and the outputs are the corresponding strong/weak labels.

**Testing:** Given a test input, we first extract its feature vector by feeding the test image to the DNN under test and then feed the extracted feature vector into trained DEEPROBUST-W, which predicts if the input image is a strong or weak point.

3.3.2 DEEPROBUST-B: Black-box Classifier

This is also a binary classifier that is intended to classify an image to strong/weak point. However, here the user does not have white box access to the DNN under test. Figure 4 shows the workflow.

Given a test input, we first randomly generate some of its neighbors. We then query the DNN under test with all these neighbors and compute the diversity score, as per Equation 1. If the neighbor diversity score (inversely correlated with neighbor diversity) is greater than a given diversity score threshold, the given test input is classified as a strong point; otherwise, a weak point.

Notice that, in this method, we do not need a training step. We only need the diversity score threshold, which can be empirically set using a ground-truth data set. In particular, we first calculate the neighbor accuracy and diversity score of each pre-annotated point. Next, based on a given neighbor accuracy threshold, we identify the weak points, as the ground truth. The highest diversity score among these weak points is chosen as the diversity score threshold.

3.3.3 Usage Scenario

DEEPROBUST-W/B works in a real-world setting where a customer/user runs a pre-trained DNN model in real-time which constantly receives inputs and wants to test if the prediction of the DNN on a given input can be trusted. DEEPROBUST-W assumes that the user has white-box access to DNN under test and all the training data used to train the DNN. DEEPROBUST-W leverages the feature vector and neighbor accuracy of the training data to train the classifier, which can notify the user if the current input is a strong point or weak point. If the input is classified as strong point, the user can give more trust to the original DNN’s prediction. On the other hand, if the point is classified as a weak point, the user may want to be more cautious about the DNN’s prediction and conduct additional inspections.

In the blackbox setting, DEEPROBUST-B assumes the user does not have white-box access to DNN under test. DEEPROBUST-B comes with a small overhead of transforming the input multiple times to get some neighbors and querying DNN under test on them to estimate the diversity score.

4 Experimental Design

4.1 Study Subjects

4.1.1 Image Classification

Similar to many existing works [4, 7, 9, 13, 35, 36] on DNN testing, in this work, we use image classification application of DNNs as the basis of our investigation. This
is one of the most popular computer vision tasks, where the model tries to classify the objects in an image or video.

**Datasets:** We conduct our experiments on three image classification datasets: F-MNIST [37], CIFAR-10 [38], and SVHN [39].

- **CIFAR-10:** consists of 50,000 training and 10,000 testing images. Each image is a 32x32 color image and is one of ten classes.
- **F-MNIST:** consists of 60,000 training images and 10,000 testing images. Each image is a 28x28 grayscale image and belongs to one of ten fashion-related classes.
- **SVHN:** consists of 73,257 training images and 26,032 testing images. Each image is a 32x32 color cropped image of house numbers collected from Google Street View images.

![TABLE 1: Study Subjects (values are in percentage)]

| Dataset | CIFAR-10 | SVHN | F-MNIST |
|---------|----------|------|---------|
| Model   | VGG | ResN | WRN | VGG | ResN | WRN | VGG | ResN | WRN |
| nat acc | 89.0 | 89.3 | 90.6 | 94.5 | 95.3 | 95.2 | 93.4 | 93.5 | 93.6 |
| rob acc* | 75.5 | 68.5 | 74.8 | 78.1 | 78.9 | 81. | 61.1 | 63.0 | 64.2 |

*Robust accuracy is estimated as the average neighbor accuracy for test data points.

**Architectures:** The popular DNN-based image classifiers are variants of convolutional neural networks (CNN) [1], [40], [41]. Here we study the following three architectures for all the three datasets:

- **ResN:** Following Engstrom et al. [18], we use ResN model with 4 groups of residual layers with filter sizes 16, 16, 32, and 64, and 5 residual units each.
- **VGG:** We use the same VGG architecture as proposed in [42].
- **WRN:** We use the structure with block type (3, 3) and depth 28 in [43] but replace the widening factor 10 with 2 for less parameters and faster training speed.

We train all the models from scratch. When training models on CIFAR-10, we pre-process the input images with random augmentation (random translation with $dx, dy \in [-2, 2]$ pixels both horizontally and vertically) which is a widely used preprocessing step for this dataset. When training models on the other two datasets, plain images are directly fed into the models. The natural accuracies and robust accuracies of the models are shown in Table 1.

### 4.1.2 Steering Angle Prediction

We further evaluate DEEPROBUST-W in a self-driving car application to show that it can be applied into a regression task. These models learn to steer (i.e., predict steering angle) by taking in visual inputs from car-mounted cameras that record the driving scene, paired with the steering angles from a human driver.

**Datasets:** We use the dataset by Stocco et al. [44], which is collected by the authors driving on three tracks of different environments in the Udacity Simulator [45]. It consists of 37888 central camera training images and 9427 central camera evaluation images. Each image is of size 320x120.

**Architectures:** We evaluate our method on the three pre-trained DNN models used in [44]: NVIDIA DAVE-2 [46], Epoch [47], and Chauffeur [48]. These models have been objects of study of many previous testing works on self-driving car [5], [6], [44].

### 4.2 Evaluation

**Evaluation Metric.** We evaluate both DEEPROBUST-W and DEEPROBUST-B for detecting weak points under twelve and nine different DNN-dataset combinations, respectively, in terms of precision, recall, and F1 score. Let us assume that $E$ is the number of weak points detected by our tool and $A$ is the number of true weak points in the ground truth set. Then the precision and recall are $\frac{|A \cap E|}{|E|}$ and $\frac{|A \cap E|}{|A|}$, respectively. F1 score is a single accuracy measure that considers both precision and recall, and defined as $\frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$.

We perform each experiment for two thresholds of neighbor accuracy that defines strong vs. weak points: 0.75 and 0.50.

**Baselines.** We compare DEEPROBUST-W and DEEPROBUST-B with two baselines. One naive baseline (denoted random) is randomly selecting the same number of points as detected by our proposed method to be weak points. Another baseline (denoted top1) is based on prediction confidence score—if the confidence of a data point is higher than a predefined cutoff we call it a strong point, weak otherwise. This baseline is based on the intuition that DNNs might not be confident enough to predict the weak points.

## 5 Results

In this section, we elaborate on our results. The first four RQs are based on the controlled study, where we spatially transformed the images with varying degrees of spatial transformations for analyzing image classification models. RQ1 and RQ2 present our empirical findings on the different characteristics of local robustness. In RQ3 and RQ4, we evaluate DEEPROBUST-W and DEEPROBUST-B in identifying the weak points. In RQ5, we evaluate the generalizability of our method by applying DEEPROBUST-W in a regression task for self-driving cars under more complex transformations.

**RQ1. How frequent are the non-robust, a.k.a. weak, points within a dataset and across different DNN models?**

We first explore the distribution of spatial robustness of the studied data sets w.r.t. three well-trained models. In particular, we explore the variations of spatial robustness: a) across different data points of a given model, b) across different models of a given data point.

#Testing Data

![Figure 5: Histogram of neighbor accuracy for CIFAR-10](image)

RQ1a. Given a well trained model, how spatial robustness changes across different data points? Here, we explore the spatial robustness of each point as captured by its neighbor
accuracy. To have a fine-granular understanding, we draw the distribution of neighbor accuracy of data points in both training set and testing set of CIFAR-10 when using VGG, ResN and WRN models, respectively, as shown in Figure [5]. The histograms show that for all these models, the neighbor accuracy vary widely across data points—most of the data points have very high neighbor accuracy, there is a non-trivial number of points having relatively low neighbor accuracy (40% of training data and 42% of testing data have neighbor accuracy < 0.75). We call such low accuracy points to be weak or non-robust points. Similarly, for neighbor accuracy < 0.50, around 16% of training data and 20% of testing data turned out to be weak. These points degrade the aggregated spatial robustness of the model. The same finding also holds for the other two datasets.

The distribution of neighbor accuracy on the training data (not shown in Figure [3]) looks similar to that for the testing data. Table 2 further shows that correlation of the number of points in each neighbor accuracy bin between training and testing data is very strong (0.999 to 0.944) with statistical significance (p < 0.05). Such strong correlation is perhaps not surprising, as training and testing data are taken from the same data distributions.

Table 2: Correlation of spatial robustness (i.e., the number of points per neighbor accuracy bin) between training and testing data

| Dataset     | CIFAR-10 | SVHN | F-MNIST |
|-------------|----------|------|---------|
| Model       | VGG      | ResN | WRN     |
| Spearman    | 0.944    | 0.979| 0.970   |
| Pearson     | 0.988    | 0.997| 0.997   |
| Spearman    | 0.998    | 0.966| 0.987   |
| Pearson     | 0.995    | 0.975| 0.995   |

Result 1: A non-trivial number of data points (46% at neighbor accuracy < 0.75) of the whole data set are weak. Robustness seems to be an intrinsic property of the data points and does not vary across models.

We further examine the per-point variation of robustness across models by analyzing how neighbor accuracy of each data point varies across models. Figure 6 shows that the change is minimal: for CIFAR-10, F-MNIST and SVHN, 60%, 76%, and 81% of data points have neighbor accuracy change less than 0.2 across the two models. The other datasets also show similar results. This implies that a large portion of data points’ neighbor accuracy is independent of the model selected. Note that data points that are non-robust across all the models will not be identified by differential testing techniques as proposed by DeepXplore [5].

RQ1b. Given a data point, how spatial robustness changes across different models? Figure 5 shows that the distribution of neighbor accuracy for a dataset looks similar across different models, indicating robustness may be a property of the dataset itself and may not depend on the underlying model architecture. To dig further, for a given dataset, we divide the neighbor accuracy of all data points into 20 equal-sized bins and compute the Spearman and Pearson correlations of the number of points falling into every bin between any two models. To reduce the potential bias induced by the large portion of original points having neighbor accuracy higher than 95%, we also compute the Pearson correlation without them. Table 3 shows that the correlations are very high between any two models for all the three datasets confirming that the overall robustness distribution of a dataset is very similar across different models.

Table 3: Correlation between models in number of points per neighbor accuracy bin

| Dataset     | Setting | Data | Spearman | Pearson | Pearson (w/o last bin) |
|-------------|---------|------|----------|---------|------------------------|
| CIFAR-10    | VGG     | ResN | 0.980    | 0.985   | 0.938                  |
|             | Test    | Test | 0.956    | 0.980   | 0.920                  |
|             | VGG     | WRN  | 0.995    | 0.996   | 0.986                  |
|             | Test    | Test | 0.977    | 0.995   | 0.980                  |
|             | ResN    | WRN  | 0.991    | 0.990   | 0.980                  |
|             | Test    | Test | 0.989    | 0.986   | 0.977                  |
| SVHN        | VGG     | WRN  | 0.955    | 0.978   | 0.997                  |
|             | Test    | Test | 0.995    | 0.983   | 0.995                  |
|             | ResN    | WRN  | 0.992    | 0.986   | 0.995                  |
|             | Test    | Test | 0.998    | 0.995   | 0.981                  |
| F-MNIST     | VGG     | ResN | 0.863    | 0.637   | 0.969                  |
|             | Test    | Test | 0.890    | 0.627   | 0.970                  |
|             | VGG     | WRN  | 0.877    | 0.883   | 0.986                  |
|             | Test    | Test | 0.893    | 0.874   | 0.969                  |
|             | ResN    | WRN  | 0.967    | 0.901   | 0.965                  |
|             | Test    | Test | 0.979    | 0.899   | 0.979                  |

Result 2: What are the characteristics of the weak points? Here we explore the characteristics of robust vs. non-robust points in their feature space. In particular, we check the difference in feature representations between: a) robust and non-robust points, and b) points with different degrees of robustness.

RQ2a. Given a well trained model, do the feature representations of robust and non-robust points vary? In
this RQ, we first explore how robust (i.e., strong) and non-robust (i.e., weak) data points are distributed in the feature space. We apply t-SNE[49], a widely used visualization method, to visualize the distribution of points of different neighbor accuracy in the representation space for all three datasets when using ResN as the classifier. Figure 7 shows the visualization of feature vectors from two randomly picked classes with colors indicating the neighbor accuracy of each point. The darker a point’s color is, the lower its neighbor accuracy is. It is evident that most points of low neighbor accuracy tend to be further away from the class center.

To numerically verify this observation, first, we define a class center \( c_k \) for each class \( k \) as the median value of the feature vectors of all the points from class \( k \). Thus, if \( f_i \) is the feature of a point at \( i^{th} \) dimension and \( f_{ik} \) is the median of the \( i^{th} \) dimension features for all the points in class \( k \), \( c_k \) is defined to be \( \{f_{1k}, \ldots, f_{jk}, \ldots, f_{nk}\} \).

The reason we take median rather than mean is that it is a more statistically stable measure and is less likely to be heavily influenced by outliers in the representation space. Then, for every point \( p \), we define a ratio:

\[
r(p) = \frac{d(p)_{same\_class}}{d(p)_{nearest\_other\_class}},
\]

where \( d(p)_{same\_class} \) is the distance of the \( p \)-th point’s feature vector to its own class center and \( d(p)_{nearest\_other\_class} \) is the distance of the \( p \)-th point’s feature vector to the class center of its closest other class. A small \( r(p) \) means that the point \( p \) is close to its own class center while far from other classes, i.e., \( p \) is far from the decision boundary. In contrast, a larger \( r(p) \) indicates that the point \( p \) is closer to some other classes, i.e., it is closer to the decision boundary.

We then measure the average \( r(p) \) among the weak points (denoted as \( r_w \)) and among strong points (denoted as \( r_s \)) for all three datasets across three models. Besides, we also calculate mann-whitney wilcoxon test[50] and cohen’s d effect size[51] between the two ratios to test if the two ratios indeed have statistically significant difference and how large the difference is.

TABLE 4: Weak and strong points ratio, and cohen’s d effect size

| Dataset | CIFAR-10 | SVHN | F-MNIST |
|---------|----------|------|---------|
| Model   | ResN WRN VGG | ResN WRN VGG | ResN WRN VGG |

| Neighbor Accuracy Cutoff=0.5 |
|-----------------------------|
| \( r_w \) | 0.915 | 0.955 | 1.004 | 1.046 | 1.103 | 1.097 | 0.746 | 0.734 | 0.976 |
| \( r_s \) | 0.609 | 0.584 | 0.975 | 0.294 | 0.309 | 0.977 | 0.297 | 0.293 | 0.930 |
| \( d^* \) | 1.368 | 1.736 | 1.163 | 2.077 | 2.428 | 1.420 | 1.426 | 1.312 | 1.332 |

| Neighbor Accuracy Cutoff=0.75 |
|-----------------------------|
| \( r_w \) | 0.778 | 0.796 | 0.992 | 0.604 | 0.671 | 0.987 | 0.516 | 0.496 | 0.953 |
| \( r_s \) | 0.588 | 0.558 | 0.973 | 0.260 | 0.274 | 0.977 | 0.253 | 0.257 | 0.918 |
| \( d^* \) | 0.786 | 1.040 | 0.749 | 0.860 | 1.111 | 0.401 | 0.749 | 0.642 | 0.937 |

Cohen’s d effect size of 0.20 means small, 0.50 means medium, 0.80 means large, 1.20 means very large, and 2.0 means huge[51, 22].

As shown in Table 4 for both the neighbor accuracy cutoff (0.5 and 0.75), except one setting, the cohen’s d effect size for every setting is larger than 0.50, which implies a medium to very large difference. Besides, for every setting, the mann-whitney wilcoxon test value (not shown in the table) is smaller than \( 1e^{-30} \), which implies the difference is indeed statistically significant.

The visualization and numerical results imply that most weak points are close to the decision boundaries between classes. Note that similar observation was also observed by Kim et al. [2] in case of adversarial perturbation. In particular, they find that adversarial examples tend to be closer to class decision boundaries. In contrast, we focus on spatial robustness and find that spatially non-robust points are closer to decision boundaries.

RQ2b. Given a well trained model, do the feature representations of the data points vary by their degree of robustness? By analyzing the classifications of the neighbors of weak vs. strong points, we observe that the weaker a point is, its neighbors are more likely to be classified in different classes. We quantify this observation by computing diversity of the outputs a point’s neighbor; We adopt Simpson Diversity Index (\( \lambda \)) as defined in Equation 1.

TABLE 5: Spearman Correlation between Neighbor Accuracy and Simpson Diversity Index. All coefficients are reported with statistical significance (\( p < 0.05 \)).

| Dataset | CIFAR-10 | SVHN | F-MNIST |
|---------|----------|------|---------|
| Model   | ResN WRN VGG | ResN WRN VGG | ResN WRN VGG |
| corr.coeff. | 0.853 | 0.909 | 0.946 | 0.970 | 0.984 | 0.983 | 0.923 | 0.962 | 0.8947 |

Table 5 shows the Spearman correlation between neighbor accuracy and \( \lambda \) on the three datasets and three models for each. Note that while calculating the correlation, we remove points with neighbor accuracy 100% since there are many points having 100% neighbor accuracy as shown in RQ1 and tend to bias upward the Spearman Correlation; if we include points with neighbor accuracy 100%, the correlations become even higher. We notice that for any setting, the Spearman Correlation is never lower than 0.853. This indicates that neighbor accuracy and diversity are highly correlated with each other. For example, the bird image in Fig.1a has neighbor accuracy 0.49 and diversity 0.36, while the bird image in Fig.1c has neighbor accuracy 1 and diversity 1. This shows, the classifier tends to be confused about weak points and mispredicts them into many different kinds of classes.

RQ3. Can we detect the weak points in a white-box setting?

We explore this RQ using DEEPROBUST-W, as discussed in Section 3.3.1. DEEPROBUST-W takes the feature vector of a data point as input and classifies it to a strong/weak point. We implement DEEPROBUST-W with a simple 4-layer, fully connected neural network architecture with hidden layer dimensions 1500, 1000, and 500, respectively.

Table 6 shows the result. At 0.75 setting, DEEPROBUST-W has \( f_1 \) up to 91.4%, with an average of 76.9%. At 0.50 setting, DEEPROBUST-W detects weak points with average \( f_1 \) of 61.1%, while it can go as high as 79.1%. DEEPROBUST-W consistently performs significantly better than the top1 baseline and random baseline. The top1 has very good
precision, since a mis-classified image with low confidence tends to have very poor local robustness. However, there also exist many images that are correctly classified with high confidence yet have poor local robustness. The miss of these points leads the top1 to have very poor recall and thus even worse f1 compared with the random baseline. Our method comes to aid by providing high recall at the same time of decent precision.

TABLE 6: Performance of DeepRobust-W and the random baseline method for predicting weak points under different settings

| dataset | model | method | 0.75 neighbor acc. | 0.50 neighbor acc. |
|---------|-------|--------|--------------------|--------------------|
| CIFAR-10 | ResN | ours | 0.79 | 3844 |
| | | top1 random | 0.747 | 2901 |
| | | ours | 0.35 | 889 |
| | | top1 random | 0.395 | 1534 |
| | | ours | 0.654 | 2222 |
| | | top1 random | 0.439 | 1070 |
| | | ours | 0.332 | 1127 |
| SVHN | ResN | ours | 0.755 | 6814 |
| | | top1 random | 0.315 | 1665 |
| | | ours | 0.343 | 3095 |
| | | top1 random | 0.332 | 1127 |
| | | ours | 0.709 | 5062 |
| | | top1 random | 0.292 | 1238 |
| | | ours | 0.595 | 5214 |
| | | top1 random | 0.172 | 840 |
| | | ours | 0.914 | 6034 |
| | | top1 random | 0.124 | 428 |
| | | ours | 0.896 | 5743 |
| | | top1 random | 0.144 | 490 |
| | | ours | 0.864 | 6348 |
| F-MNIST | ResN | ours | 0.709 | 5062 |
| | | top1 random | 0.292 | 1238 |
| | | ours | 0.595 | 5214 |
| | | top1 random | 0.172 | 840 |
| | | ours | 0.914 | 6034 |
| | | top1 random | 0.124 | 428 |
| | | ours | 0.896 | 5743 |
| | | top1 random | 0.144 | 490 |
| | | ours | 0.864 | 6348 |

Notice that DeepRobust-W’s performance depends on the training data selection, mainly (a) how many weak vs. strong points are used to train the model, and (b) how many neighbors are generated per point to decide whether it is strong/weak. To investigate the previous one, we assign a weight to each input point, indicating how likely it would be selected to train DeepRobust-W. In particular, for an input \( i \), a weight \( w_i := \frac{1}{1 + (\sum_{j \in \mathcal{N}_i}^m w_j)^{\frac{1}{m}}} \) is computed, where \( m \) is its neighbor accuracy, and \( m \) is a configurable parameter; with larger \( m \), more weak points are sampled, and vice versa. Thus, if \( m \) is larger, DeepRobust-W will be trained with more weak points and vice versa. Table 6 shows the performance: as \( m \) increases, the detector trades precision for recall. In this way, choosing different values of \( m \), the precision-recall trade-off of the detector can be adjusted according to a user’s need. From a different perspective, this way of oversampling weak points also addresses the potential problem of imbalanced data when the weak points are much less than the strong points.

Next, we check how DeepRobust-W’s performance is dependent on the number of sampled neighbors, because a data point can potentially have infinite neighbors. Table 7 shows that the number of neighbors does not have much influence on the performance of the detector once it goes beyond some value (F1 score does not change more than 3.5 percentage point between 25 and 200 samples) for all the three datasets. Thus, we choose 50 for all of our experiments.

TABLE 7: DeepRobust-W performance using different sampling strategies for training

A: with varying number of strong/weak points

| Dataset | CIFAR-10 | SVHN | F-MNIST |
|---------|---------|------|---------|
| \( m \) | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 |
| prec | 0.660 | 0.615 | 0.544 | 0.677 | 0.575 | 0.332 | 0.794 | 0.746 | 0.712 |
| recall | 0.518 | 0.599 | 0.699 | 0.502 | 0.653 | 0.767 | 0.767 | 0.839 | 0.871 |
| f1 | 0.581 | 0.607 | 0.612 | 0.577 | 0.612 | 0.463 | 0.791 | 0.791 | 0.783 |
| tp | 1290 | 1490 | 1740 | 1414 | 1837 | 2160 | 2144 | 2284 | 2372 |
| fp | 664 | 932 | 1460 | 674 | 1357 | 4356 | 556 | 777 | 962 |

B: with varying number of neighbors

| Dataset | #neighbors | prec | recall | tp | fp | f1 |
|---------|------------|------|--------|----|----|----|
| CIFAR-10 | 6 | 0.662 | 0.389 | 967 | 493 | 0.49 |
| | 12 | 0.685 | 0.394 | 955 | 440 | 0.492 |
| | 25 | 0.665 | 0.502 | 1250 | 629 | 0.572 |
| | 50 | 0.660 | 0.518 | 1290 | 664 | 0.581 |
| | 200 | 0.683 | 0.507 | 1261 | 585 | 0.582 |
| SVHN | 6 | 0.723 | 0.403 | 1136 | 436 | 0.518 |
| | 25 | 0.672 | 0.527 | 1483 | 725 | 0.559 |
| | 50 | 0.632 | 0.605 | 1703 | 993 | 0.618 |
| | 200 | 0.667 | 0.550 | 1550 | 774 | 0.603 |
| F-MNIST | 6 | 0.817 | 0.727 | 1981 | 443 | 0.77 |
| | 12 | 0.784 | 0.790 | 2153 | 592 | 0.787 |
| | 25 | 0.773 | 0.787 | 2143 | 629 | 0.789 |
| | 50 | 0.836 | 0.727 | 1981 | 390 | 0.778 |
| | 200 | 0.778 | 0.812 | 2211 | 632 | 0.794 |

Result 3: DeepRobust-W can identify weak points with reasonably high F1 score: on average 76.9%, at 0.75 neighbor accuracy cut-off.

RQ4. Can we identify the weak points in a black-box setting?

We explore this RQ using DeepRobust-B, as discussed in Section 3.3.2. Here, we assume, we only have access to unlabeled testing data and black-box access of the model under test. To evaluate DeepRobust-B, we spatially transform each test input \( m \) times by randomly applying \( d \omega \in [-30, +30] \) degrees rotation, \( dx \in [-3, +3] \) pixels horizontal translation, and \( dy \in [-3, +3] \) pixels vertical translation. We then calculate the output diversity score (\( \lambda \)) based on Equation 1 and rank the test images based on \( \lambda \). Finally, we mark top \( k \) images as potential most non-robust points. The parameter \( k \) is chosen according to users’ need.

With each test data, DeepRobust-B queries the model with \( m \) neighbors to compute \( OBUST \) performance using different sam-
notice that at $m = 15$, the correlation coefficients across all the experimental settings reach above 0.8, and the rate of increase begins to slow down significantly. The results for the other two architectures are highly similar. Thus, we set $m = 15$ as default for DEEPROBUST-B.

![AUC-ROC curve with neighbor accuracy cutoff at 0.75.](image)

Fig. 8: The spearman correlation coeff. between diversity score ($\lambda$) and neighbor accuracy, with varying #neighbors ($m$).

Next, we evaluate DEEPROBUST-B’s performance. We plot AUC-ROC by changing $top - k$ at $m = 15$ and compare our method with the random baseline and the top1 baseline as before. As shown in Figure 9, our method performs much better than the random baseline. In particular, our proposed method achieves AUC higher than 0.87 for all settings when neighbor accuracy cutoff is 0.5 and 0.97 when neighbor accuracy cutoff is 0.75.

![AUC-ROC curve with neighbor accuracy cutoff at 0.75.](image)

Fig. 9: AUC-ROC curve with neighbor accuracy cutoff at 0.75. The red vertical line indicates the false- and true-positive rate when the diversity score threshold is chosen from training data.

Instead of above ranking based scheme, DEEPROBUST-B can also be used as a classifier if a diversity threshold is given (see Section 3.3.2). Here, we estimate the threshold using pre-annotated training data. We evaluate precision and recall of DEEPROBUST-B in the nine DNN-dataset combinations under neighbor accuracy cutoffs 0.5 and 0.75. Table 8 shows the result. At 0.75 setting, DEEPROBUST-B has f1 up to 99.1%, with an average of 96.5%. At 0.50 setting, DEEPROBUST-B detects weak points with average f1 of 72.9%, while it can go as high as 85.7%. It consistently produces much better estimation than the top1 baseline and the random baseline. This shows that our black-box method can effectively identify weak points.

| dataset | model    | method | 75%  | 50%  |
|---------|----------|--------|------|------|
|        |          | f1   | tp  | fp  | f1   | tp  | fp  |
| CIFAR-10 | ResN    | ours | 0.939 | 4714 | 257 | 0.622 | 1454 | 801 |
|         |         | top1 | 0.376 | 1218 | 206 | 0.182 | 255  | 120 |
|         |         | random | 0.501 | 2516 | 2455 | 0.234 | 549  | 1706 |
| WRN     | ours    | 0.938 | 3657 | 171 | 0.585 | 986  | 604  |
|         | top1    | 0.35  | 889  | 222 | 0.183 | 189  | 90   |
|         | random  | 0.383 | 1494 | 2334 | 0.182 | 307  | 1283 |
| VGG     | ours    | 0.945 | 3397 | 148 | 0.682 | 1087 | 390  |
|         | top1    | 0.439 | 1070 | 153 | 0.266 | 278  | 106  |
|         | random  | 0.36  | 1296 | 2249 | 0.153 | 244  | 1233 |
| SVHN    | ResN    | ours | 0.956 | 8371 | 365 | 0.67 | 1845 | 858 |
|         |         | top1 | 0.315 | 1665 | 142 | 0.267 | 452  | 122 |
|         |         | random | 0.336 | 2944 | 5792 | 0.102 | 280  | 2423 |
| WRN     | ours    | 0.963 | 6827 | 227 | 0.718 | 1602 | 514  |
|         | top1    | 0.292 | 1238 | 130 | 0.203 | 275  | 85   |
|         | random  | 0.275 | 1950 | 5104 | 0.085 | 191  | 1925 |
| VGG     | ours    | 0.976 | 8608 | 144 | 0.779 | 2138 | 454  |
|         | top1    | 0.172 | 840  | 67  | 0.139 | 221  | 52   |
|         | random  | 0.339 | 2997 | 5755 | 0.102 | 279  | 2313 |
| F-MNIST | ResN    | ours | 0.987 | 6422 | 81 | 0.802 | 2316 | 546 |
|         |         | top1 | 0.124 | 428  | 11 | 0.039 | 57   | 7    |
|         |         | random | 0.655 | 4265 | 2238 | 0.289 | 835  | 2027 |
| WRN     | ours    | 0.989 | 6246 | 70  | 0.857 | 2297 | 360  |
|         | top1    | 0.144 | 490  | 14  | 0.045 | 63   | 8    |
|         | random  | 0.631 | 3987 | 2329 | 0.274 | 736  | 1921 |
| VGG     | ours    | 0.991 | 7078 | 60  | 0.847 | 2393 | 418  |
|         | top1    | 0.104 | 392  | 5   | 0.028 | 39   | 5    |
|         | random  | 0.711 | 5084 | 2054 | 0.277 | 784  | 2027 |

Note that, generating the spatial transformations and querying the model with it under black box setting is fast. Previous black box methods for adversarial perturbation work in such fashion [53], [54]. For example, using CIFAR-10, when we use a batch with size 100, the average transformation-query time for one image is $0.031 \pm 0.015$ ms. For the other two datasets, the overhead is similar. Thus, to for $m = 15$ queries, it takes only $0.465 \pm 0.225$ ms, which is a negligible overhead for most real-world DNN based vision applications. This implies that our black-box method can also be used in real time for many applications.

Result 4: Given only black-box access to the DNN classifier, DEEPROBUST-B can identify weak points with f1 that are much better than those of using top1 method or random method.

RQ5. How generalizable are these findings?

The local robustness issues also exist in more critical applications like self-driving-car. Here we explore more complex transformations, i.e., adding rain and fog to the driving scenes. As shown in Figure 10 among those correctly classified data points, there is a non-trivial portion
try different variations of an image in real-time for a self-driving car, which is a time-sensitive application. Further, DeepRobust-B requires the calculation of neighbor diversity score. For a regression problem, the predicted values are continuous, so there is a very low probability for any two predictions being equal. Thus, the neighbor diversity score for every data point will be the same and cannot be used for identifying the weak points.

Result 5: DeepRobust-W can detect weak points of a self-driving car dataset with f1 score up to 78.9%, with an average of 58.2%, at neighbor accuracy cutoff 0.75.

6 RELATED WORK

Adversarial examples. Many recent works focus on generating adversarial examples to fool the DNNs and evaluate their robustness using pixel-based perturbation [17, 20]. This includes different techniques to automatically distinguish adversarially modified inputs from natural ones [53, 56], to verify the robustness of DNNs to adversarial inputs [57–59], techniques to harden DNNs against adversarial attack [60–62], and techniques to analyze the general properties of adversarial inputs [63, 64]. Some other papers [18, 32, 33], like us, proposed more realistic transformations to generate adversarial examples. In particular, Engstrom et al. [18] proposed that a simple rotation and translation can fool a DNN classifier, and spatial adversarial robustness is orthogonal to \( l_p \)-bounded adversarial robustness. However, all these works estimate the overall robustness of a DNN based on its aggregated behavior across many data points. In contrast, we assume that not all the data points in a dataset have similar robustness properties. Thus, we analyze the robustness of individual data points under natural variations and propose methods to detect weak/strong points automatically.

DNN testing. Many researchers [9, 15, 35, 65–69] proposed techniques to test DNN. For example, Pei et al. [5] proposed an image transformation based differential testing framework, which can detect erroneous behavior by comparing the outputs of an input image across multiple DNNs. In contrast, others [6, 7, 66, 69] used metamorphic testing where the assumption is the outputs of an original, and its transformed image will be the same under natural transformations. We follow a similar metamorphic property while estimating neighbor accuracy. Another line of work [70–73] estimates the confidence of a DNN’s output [74, 75]. For example, [71] leverages thrown away information from existing models to measure confidence; [72] shows other NN properties like depth, width, weight decay, and Batch Normalization are important factors influencing prediction confidence. Although such methods can provide a confidence measure per input, it does not check its robustness property, i.e., with variations how will they behave.

DNN verification. Existing works [57, 75–79] on verification of properties for a DNN model focus on verifying properties on \( l_p \) norm bounded input space. Recently, Balunovic et al. [22] provides the first verification technique for verifying a data point’s robustness against spatial transformation. However, their technique suffers from scalability issues. The proposed method has only been shown to work
for small transformation (rotation with at most 10 degrees) on small architectures (CNN with at most 4 layers). Even when the tolerance of verification is low, their average verification time is 1 s/image. In contrast, our approach can evaluate a data point’s robustness against more natural transformations (e.g. rain, fog), at a larger transformation (e.g. ± 30 degrees for rotation), scale to large architectures (e.g. WRN), and run much faster (1 ms/image on average). Thus, our testing method serves as a good complement.

Robust training. Regular neural network training involves the optimization of the loss value for each data point. Robust training of neural network works on minimizing the largest loss within a specific bounded region usually using adversarial examples [32], [57], [59], [80]–[85]. While both robust training methods and our work generate variants of data points, instead of training a model with these variants to improve robustness, we leverage them to estimate the robustness of the unseen data points. The relation between robust retraining and us is thus similar to bug fixing vs. bug detection in traditional software engineering literature.

7 Threats to Validity

We adopt rotation and translation as transformations for image classification tasks and rain and fog effects for the self-driving car task. There are many more natural variations such as brightness, snow effect etc. However, rotation and translation are representative of spatial transformation and used by many paper in evaluating robustness of DNN models [5], [18]. Rain and fog effects are also widely leveraged in many influential studies on testing self-driving cars [5]–[7].

Besides, for some of the experiments we did not show all the combinations under both neighbor accuracy cutoffs (i.e. 0.5 and 0.75). However, we note that the observations are consistent and we did not include them purely because of space limitation. Another limitation is that for both DeepRobust-W and DeepRobust-B, we need to decide the number of neighbors to use for training a classifier and estimating λ, respectively. We mitigate it by selecting the neighbor numbers that give stable performance in terms of precision and recall.

8 Conclusion

In this work, we involve the data characteristic into the robustness testing of a DNN model. We adopt the concept of neighbor accuracy as a measure for local robustness of a data point on a given model. We explore the properties of neighbor accuracy and find that it is a relatively stable measure across models for most data points and thus, inherent to the dataset rather than the model used. We also observe that weak points are often located towards the corresponding class boundaries and their transformed versions are likely to be predicted to be more diverse classes. By leveraging these two observations, we propose a white-box method and a black-box method to identify weak/strong points to warn a user about potential weakness in the given trained model in real-time. We design, implement and evaluate our proposed framework, DeepRobust, on three image recognition datasets and one self-driving car dataset with three models for each. The results show that DeepRobust is able to identify weak points and strong points with high precision and recall. Besides, it allows a user to trade off precision for recall to accommodate different needs.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in Advances in neural information processing systems, 2012, pp. 1097–1105.
[2] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Montfort, U. Muller, J. Zhang et al., “End to end learning for self-driving cars,” arXiv preprint arXiv:1604.07316, 2016.
[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, “Mastering the game of go with deep neural networks and tree search,” Nature, vol. 529, pp. 484–503, 2016. [Online]. Available: http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html
[4] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, “Software engineering for machine learning: A case study,” in Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, ser. ICSE-SEIP ‘19. IEEE Press, 2019, pp. 291–300. [Online]. Available: https://doi.org/10.1109/ICSE-SEIP.2019.00042
[5] K. Pei, Y. Cao, J. Yang, and S. Jana, “Deepxplore: Automated whitebox testing of deep learning systems,” in Proceedings of the 26th Symposium on Operating Systems Principles. ACM, 2017, pp. 1–18.
[6] Y. Tian, K. Pei, S. Jana, and B. Ray, “Deepstest: Automated testing of deep-neural-network-driven autonomous cars,” in International Conference on Software Engineering (ICSE), 2018 IEEE conference on. IEEE, 2018.
[7] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, “Deeproad: Gan-based metamorphic autonomous driving system testing,” arXiv preprint arXiv:1802.02285, 2018.
[8] S. Ma, Y. Liu, W.-C. Lee, X. Zhang, and A. Grama, “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. ACM, 2018, pp. 175–186.
[9] J. Kim, R. Feldt, and S. Yoo, “Guiding deep learning system testing using surprise adequacy,” in Proceedings of the 41st International Conference on Software Engineering. IEEE Press, 2019, pp. 1039–1049.
[10] H. V. Pham, T. Lutellier, W. Qi, and L. Tan, “Cradle: Cross-backend validation to detect and localize bugs in deep learning libraries,” in Proceedings of the 41st International Conference on Software Engineering, ser. ICSE ’19. IEEE Press, 2019, p. 1027–1038. [Online]. Available: https://doi.org/10.1109/ICSE.2019.00107
[11] X. Du, X. Xie, Y. Li, L. Ma, Y. Liu, and J. Zhao, “Deepstellar: Model-based quantitative analysis of stateful deep learning systems,” in Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ser. ESEC/FSE ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 477–487. [Online]. Available: https://doi.org/10.1145/3338906.3338954
[12] M. J. Islam, G. Nguyen, R. Pan, and H. Rajan, “A comprehensive study on deep learning bug characteristics,” in Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ser. ESEC/FSE ’19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 510–520. [Online]. Available: https://doi.org/10.1145/3338906.3338955
[13] Z. Li, X. Ma, C. Xu, C. Cao, J. Xu, and J. Lu, “Boosting operational dmn testing efficiency through conditioning,” in Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ser. ESEC/FSE ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 499–509. [Online]. Available: https://doi.org/10.1145/3338906.3338930
[14] H. Zhang and W. K. Chan, “Apricot: A weight-adaptation approach to fixing deep learning models,” in 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), Nov 2019, pp. 376–387.

[15] Y. Tian, Z. Zhong, Y. Ordonez, G. Kaiser, and B. Ray, “Testing deep image classifiers for confusion & bias errors,” in International Conference of Software Engineering (ICSE), 2020.

[16] X. Gao, R. Saha, M. Prasad, and A. Roychoudhury, “Fuzz testing based data augmentation to improve robustness of deep neural networks,” in Proceedings of the 42nd International Conference on Software Engineering, ser. ICSE 2020. ACM, 2020.

[17] L. Engstrom, B. Tran, D. Tsipras, L. Schmidt, and A. Madry, “Explaining and harnessing adversarial examples,” in International Conference on Learning Representations (ICLR), 2015.

[18] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, “End to end learning for self-driving cars.” CoRR, vol. abs/1604.07316, 2016. [Online]. Available: http://arxiv.org/abs/1604.07316

[19] Y. Tian, Z. Zhong, V. Ordonez, G. Kaiser, and B. Ray, “Testing adversarial examples with adversarial networks,” in International Conference on Learning Representations (ICLR), 2018.

[20] L. Engstrom, B. Tran, D. Tsipras, L. Schmidt, and A. Madry, “Robustness may be at odds with accuracy,” in International Conference on Learning Representations (ICLR), 2019.

[21] S. Wang, Y. Chen, A. Abdou, and S. Jana, “Mixtrain: Scalable training of formally robust neural networks,” arXiv preprint arXiv:1811.02625, 2018.

[22] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards deep learning models resistant to adversarial attacks,” in International Conference on Learning Representations (ICLR), 2017.

[23] X. Gao, R. Saha, M. Prasad, and A. Roychoudhury, “Fuzz testing based data augmentation to improve robustness of deep neural networks,” in Proceedings of the 42nd International Conference on Software Engineering, ser. ICSE 2020. ACM, 2020, p. 12 pages.

[24] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, “End to end learning for self-driving cars.” CoRR, vol. abs/1604.07316, 2016. [Online]. Available: http://arxiv.org/abs/1604.07316

[25] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and A. C. Courville, “Deep learning,” https://arxiv.org/abs/1412.6550, 2014.

[26] S. Bengio, J. Louradour, R. Collobert, and J. Weston, “Conservation of weights and gradients in deep learning,” in Proceedings of the 31st international conference on machine learning (ICML), 2014.

[27] A. Krizhevsky, A. Sutskever, and I. Goodfellow, “ImageNet classification with deep convolutional neural networks,” in Proceedings of the 25th international conference on machine learning, ser. ICML ’09. ACM, 2009, pp. 386–399. [Online]. Available: http://dl.acm.org/citation.cfm?id=163688a0

[28] S. Bengio, J. Louradour, R. Collobert, and J. Weston, “Conservation of weights and gradients in deep learning,” in Proceedings of the 31st international conference on machine learning (ICML), 2014.

[29] L. van der Maaten and G. Hinton, “Visualizing data using t-SNE,” Journal of Machine Learning Research, vol. 9, pp. 2579–2605, 2008. [Online]. Available: http://www.jmlr.org/papers/v9/vandermaaten08a.html

[30] T. W. Yuval Netzer, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, “Reading digits in natural images with unsupervised feature learning,” in NIPS Workshop on Deep Learning and Unsupervised Feature Learning, 2011.

[31] 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.

[32] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and A. Courville, “Deep learning,” https://arxiv.org/abs/1412.6550, 2014.

[33] A. Krizhevsky, A. Sutskever, and I. Goodfellow, “ImageNet classification with deep convolutional neural networks,” in Proceedings of the 25th international conference on machine learning (ICML), 2009.

[34] M. Balunovic, M. Baader, G. Singh, T. Gehr, and M. Vechev, “Certifying geometric robustness of neural networks,” in Advances in Neural Information Processing Systems, 2019, pp. 15287–15297.

[35] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards deep learning models resistant to adversarial attacks,” in International Conference on Learning Representations (ICLR), 2018.

[36] C. Xiao, B. Li, J.-Y. Zhu, W. He, M. Liu, and D. Song, “Generating adversarial examples with adversarial networks,” in 27th International Joint Conference on Artificial Intelligence (IJCAI), 2018.

[37] L. Engstrom, B. Tran, D. Tsipras, L. Schmidt, and A. Madry, “A rotation and a translation suffix: Fooling cnns with simple transformations,” arXiv preprint arXiv:1712.02779, 2017.

[38] J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in Advances in neural information processing systems, 2014, pp. 2672–2680.

[39] Z. Zhao, D. Du, and S. Singh, “Generating natural adversarial examples,” in International Conference on Learning Representations (ICLR), 2018.

[40] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry, “Robustness may be at odds with accuracy,” in International Conference on Learning Representations (ICLR), 2019.

[41] E. H. Simpson, “Measurement of diversity,” Nature, vol. 163, no. 4148, pp. 688–689, 1949. [Online]. Available: https://doi.org/10.1038/163688a0

[42] L. Engstrom, B. Tran, D. Tsipras, L. Schmidt, and A. Madry, “A rotation and a translation suffix: Fooling cnns with simple transformations,” in Proceedings of the 36th international conference on machine learning (ICML), 2019.

[43] C. Xiao, J.-Y. Zhu, B. Li, W. He, M. Liu, and D. Song, “Spatially transformed adversarial examples,” in International Conference on Learning Representations (ICLR), 2018.

[44] U. Saxena, “Automold,” https://github.com/UjjwalSaxena/Automold--Road-Augmentation-Library

[45] L. Ma, F. Juefei-Xu, J. Sun, C. Chen, T. Su, F. Zhang, M. Xue, B. Li, L. Li, Y. Liu et al., “Deepgage: Comprehensive and multi-granularity testing criteria for gauging the robustness of deep learning systems,” arXiv preprint arXiv:1803.07519, 2018.

[46] K. Sen, D. Marinov, and G. Agha, “CUTE: A concord unit testing environment for C,” IEEE TSE, 2015.

[47] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms,” 2017.

[48] A. Krizhevsky, “Learning multiple layers of features from tiny images,” University of Toronto, 05 2012.
