2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)

Deep Validation: Toward Detecting Real-world Corner Cases for Deep Neural Networks

Weibin Wu*†, Hui Xu*†, Sanqiang Zhong*, Michael R. Lyu*†, and Irwin King*†
* Shenzhen Research Institute, The Chinese University of Hong Kong
† Department of Computer Science and Engineering, The Chinese University of Hong Kong
{wbwu, hxu, lyu, king}@cse.cuhk.edu.hk sanqiang_zhong@whu.edu.cn

***Abstract*—The exceptional performance of Deep neural networks (DNNs) encourages their deployment in safety- and dependability-critical systems. However, DNNs often demonstrate erroneous behaviors in real-world corner cases. Existing countermeasures center on improving the testing and bug-fixing practice. Unfortunately, building a bug-free DNN-based system is almost impossible currently due to its black-box nature, so anomaly detection is imperative in practice.**

Motivated by the idea of data validation in a traditional program, we propose and implement *Deep Validation*, a novel framework for detecting real-world error-inducing corner cases in a DNN-based system during runtime. We model the specifications of DNNs by resorting to their training data and cast checking input validity of DNNs as the problem of discrepancy estimation. Deep Validation achieves excellent detection results against various corner case scenarios across three popular datasets. Consequently, Deep Validation greatly complements existing efforts and is a crucial step toward building safe and dependable DNN-based systems.

***Keywords*-neural networks, classification, safety, anomaly detection**

I. Introduction

Deep neural networks (DNNs), as emerging machine learning techniques, have amazingly approached or surpassed human performance on diverse tasks, such as image classification [18], [25], [28], machine translation [17], [64], [70], and text analysis [10], [20], [43]. These advances have facilitated the application of DNNs in a growing spectrum of safety- and dependability-critical domains, like self-driving [66], [73], biometric authentication [34], [61], and medical diagnosis [35], [71].

Despite the impressive capacities, researchers recently uncover a prominent issue in the context of image classification that these top performers often exhibit unexpected behaviors facing unseen real-world corner cases. For example, a Tesla car in Autopilot mode failed to identify a trailer against a bright sky [63], and an autonomous Uber vehicle misclassified a pedestrian during road-test at night [68], both resulting in deadly accidents. In essence, real-world corner cases are naturally transformed images that will not harm human perception [57], [67]. Therefore, it is dangerous to trust the prediction of a DNN classifier when developers do not include similar scenes in its training data. We focus on coping with such accidental failures of DNN classifiers in this paper.

Existing solutions to harden DNNs against such flaws mainly follow the idea of model retraining with data augmentation [13], [28], [57], [58], [67]. They assume that the DNN classifier has never seen these difficult corner cases before, and retraining with known corner cases can contribute to a more knowledgeable model. Unfortunately, real-world scenes can vary with many factors, like brightness, camera alignment, and object movements. Hence the training data we possess are just a relatively small fraction of all scenarios in practice. Beyond that, it is also doubtful whether there exists a perfect DNN classifier that can handle all possible images in light of the "no free lunch" theorem [69], [72]. Such methods are also notorious for their painful bug-fixing process since it is computationally intensive to train DNNs which usually contain millions of parameters.

Therefore, corner case detection should be an indispensable safety tool when deploying DNNs in real-world systems. This kind of anomaly detection is a fundamental building block in many fail-safe systems. It is employed to foresee possible risks, which enables human intervention to correct system errors. However, to our best knowledge, there is *no existing detection method* in the literature tailored for addressing such real-world error-inducing corner cases in DNNs.

We believe a meaningful corner case detector should be *scenario-agnostic*. It is discouraged to build a detector upon known anomalies, which may inherit similar drawbacks of model retraining. We consider error-inducing corner cases come from distributions that the classifier has not yet learned to settle. Motivated by the idea of data validation in traditional software engineering [1], [9], [19], [60], we hence infer error-inducing corner cases are out of range of the valid input domain of a DNN model and propose *Deep Validation*. It proceeds by validating intermediate model inputs/states to identify invalid examples that may lead to misbehaviors of the whole system. As a result, the detector is favorably model-dependent rather than scenario-dependent.

Data validation within a traditional program often resorts to specific validation constraints, because the logic of a program is manually defined and is capable of being expressed as succinct control flow statements. However, a similar validation process appears infeasible for a DNN model due to the distinct design philosophy. The functionality of a DNN model is indeed learned from a large amount of training data spontaneously without much human supervision. Therefore, its knowledge is encoded in millions of indecipherable parameters as well as the associated intricate network structures [15], [50].

978-1-7281-0057-9/19/$31.00 ©2019 IEEE
DOI 10.1109/DSN.2019.00026

To tackle these challenges, we first justify why we can instead look for support from the training data for the declaration of valid inputs. We then show how to model the valid input range of intermediate layers within DNNs through characterizing reference distributions. We follow by introducing our approach to quantify the validity of input images by estimating their discrepancy to the valid input region. We finally provide extensive experiments to corroborate the high effectiveness and great superiority of our framework in detecting real-world corner cases. Along with other merits we demonstrate, Deep Validation conduces to promoting the safety and dependability of DNN-based systems.

In summary, the main contributions of this work are:

- We formally reveal the risk of real-world corner cases for DNN-based systems. To this end, we first specify the fault model of a DNN classifier to elucidate the real-world corner cases well recognized in the community (Section II-B). Next, we adopt metamorphic testing techniques to synthesize such error-inducing samples, which reflects the variable working environment of DNNs (Section III-A) [8]. We show that these real-world corner cases can significantly undermine the safety and dependability of DNN-based systems (Section IV-B).
- We introduce Deep Validation as the *first framework* to automatically validate internal inputs/states and identify error-inducing real-world corner cases for a working DNN-based system. It monitors the deviation from the normal functionality of internal components within a DNN and makes sure this black-box system works correctly. As such, Deep Validation contributes to further improving the safety and dependability of a DNN-based system by enabling fail-safe solutions (Section III-B).
- We conduct extensive experiments across various datasets and DNN architectures to evaluate our framework. Deep Validation consistently reports prominent results on eight different categories of corner cases with an overall ROC-AUC score of 0.9937 on MNIST, 0.9805 on CIFAR-10, and 0.9506 on SVHN, respectively (Section IV-D3). The superior performance of Deep Validation also breaks the unexplored belief that detection methods tailored for intentional attacks can also work well facing real-world corner cases (Section IV-D4).
- We investigate the efficacy of Deep Validation on defending against numerous cutting-edge white-box attacks. It also achieves impressive performance with an overall ROC-AUC score of 0.9572 and 0.9755 in two settings respectively. Both are competitive with state-of-the-art results (Section IV-D5). We further test Deep Validation under high dynamical range working conditions. It confirms that Deep Validation is sensitive to impending dangers with consistently satisfactory detection rates (Section IV-D6).

II. Preliminaries

In order to precisely characterize the real-world corner cases that attract substantial interest in the community and disclose their harmfulness, we first elaborate on the system model and fault model of DNN classifiers considered in this work [57], [58], [67]. We then present a categorization of representative techniques for adversarial image detection, which makes it convenient to determine the state-of-the-art benchmark methods for better comparisons.

A. System Model of DNN Classifiers

In the field of image classification, a customized DNN structure, convolutional neural network (CNN), is embraced as the canonical solution [22], [30]. CNN classifiers often stack numerous simple components (namely, layers of neurons) with non-linear activation. These simple components collaborate to extract increasingly abstract features automatically [21], [47], [48], [76]. As a whole, they can learn the mapping between raw input images and a predefined set of labels (namely, classes of images) by mere end-to-end supervision.

In this sense, we can regard a CNN as a function $f : X \to Y$. Here X denotes the input space, and Y is the output space representing a predefined categorical set $\{1, ..., N\}$. In a conventional CNN, neurons in each layer are connected to the ones in the succeeding layer by weighted edges. We can thus regard the i^{th} layer (with $i \in \{1, ..., L\}$) as a function $f_i(f_{i-1}; \theta_i)$. It is controlled through parameters θ_i and takes as inputs the outputs from the former layer f_{i-1}.

Consequently, we can formulate the link between input images and label predictions delineated by a CNN as a composition of L parametric functions:

$$f(\mathbf{x}; \theta) = f_L(f_{L-1}(...f_1(\mathbf{x}; \theta_1); \theta_{L-1}); \theta_L). \tag{1}$$

Here $\mathbf{x}$ represents the vectorized pixel values of raw images, which is considerably high-dimensional. The parameter set $\theta := \{\theta_1, \theta_2, ..., \theta_L\}$ hence encodes the model knowledge learned from the training data. When it is clear from the context, we may take $f_i(\mathbf{x})$ or f_i as shorthand for the output of the i^{th} layer for the input image $\mathbf{x}$.

We usually translate the final output of a CNN classifier as a probability vector $f(\mathbf{x})$, where the k^{th} entry $f(\mathbf{x})[k]$ stands for the confidence of the model on categorizing image $\mathbf{x}$ as class k. Because the last layer is a softmax layer, and we can see the final output $f(\mathbf{x})$ as fitting a logistic regression on the logit outputs z_k ($k \in \{1, ..., N\}$) of the penultimate layer.

B. Fault Model of DNN Classifiers

We adopt a behavioral-level fault model to specify the fault type of DNN classifiers covered in this paper [5], [26], [59]. It clarifies the real-world corner cases with which we are concerned. Therefore the fault model can not only guide the selection of strategies for corner case generation but also profile the testbed for evaluating anomaly detection methods.

Although the defects of computing infrastructures can thwart the normal functionality of a DNN-based system, the misuse of DNN classifiers is one main culprit of their failures in practice, which remains challenging to alleviate [11], [13], [23], [49], [57], [58], [67], [77]. As introduced before, a DNN classifier merely possesses a limited capacity, and hence it

often misbehaves in the presence of unfamiliar images. In contrast with the training data of a DNN classifier, these error-inducing samples bear distinct characteristics. They usually arise out of unexpected, especially dramatic changes in the working environment, like the variations in illumination, which is the reason why they are dubbed real-world corner cases in the literature [57], [58], [63], [67], [68].

Therefore in order to simulate variable working conditions a DNN-based system may face, we follow the idea of the metamorphic testing to generate real-world corner cases [8]. Specifically, we convert available normal images into real-world corner cases by applying naturally occurring image transformations *without* destroying their original semantic meanings. For example, utilizing image rotation, we can craft images perceived by a DNN-based system when its camera deviates from the original position. We believe error-inducing corner cases are not born of the same distribution that the DNN classifier has grasped. Hence identifying these abnormal inputs beforehand can prevent the misuse of DNN classifiers.

C. Adversarial Sample Detection

Recent works on adversarial machine learning conceive a malicious context. They discover that perturbing clean images with imperceptible noise can fool state-of-the-art DNN classifiers to make wrong predictions with high confidence [7], [16], [38], [44], [45], [54], [55], [65]. As such, attackers in their thought are allowed to modify the pixel values of images freely and feed the resultant artifacts directly to the victim model. In the fight against such white-box attacks, recent studies instead turn to adversarial image detection considering the vast input space exploitable. Cutting-edge solutions mainly resort to two strategies: prediction inconsistency and statistical detection [74].

Prediction inconsistency based detection builds upon the instability of model predictions on adversarial samples. In particular, it is found that transformed adversarial images are more prone to introduce large variations into model outputs than clean counterparts. Meng and Chen [41] exploit auto-encoders to perform image transformation. In contrast, Xu *et al.* independently propose another better and cheaper technology named *feature squeezing*, where they instead filter images by different "hard-coded" squeezers [74]. They achieve exceptional results in detecting a variety of state-of-the-art white-box attacks. Nevertheless, as evaluated in Section IV-D4, this approach gives inferior detection performance when confronted with real-world corner cases.

Statistical detection generally seeks to discriminate adversarial images from clean ones through the hidden representations extracted by DNNs. Metzen *et al.* utilize the outputs from one middle layer of the target DNN to train a small subnetwork, which acts as a detector [42]. Similarly, Lu *et al.* quantize outputs from the last ReLU layer to learn a detector [36]. Ma *et al.* instead leverage all the transformation layers to characterize the Local Intrinsic Dimensionality (LID) of adversarial samples [37]. Unfortunately, detection methods of this type require both clean and adversarial training samples. Therefore, they often generalize poorly to unseen or stronger attacks [6].

Feinman *et al.* overcome this flaw by performing *kernel density estimation* to capture the statistical properties of legitimate images directly [14]. To the same end, Zheng *et al.* choose to employ Gaussian Mixture Model (GMM) approximation [78], while Lee *et al.* propose to model class conditional Gaussian distributions on the penultimate layer of DNNs [32]. They exploit only the outputs from the fully connected hidden layers to simplify the distribution they need to deal with and hope that adversarial samples can retain malicious in these layers. Pang *et al.* apply kernel density estimation to DNNs improved with reverse cross-entropy (RCE) training. They again demonstrate the outstanding performance of kernel density estimation [52]. We thus decide to adopt kernel density estimation as a representative baseline under this category. Despite achieving fabulous results in spotting adversarial images, the comparison between kernel density estimation and ours in Section IV-D4 reveals their limitations in addressing real-world corner cases.

III. Methodology

A. Corner Case Generation

In line with the fault model we account for, we now detail the metamorphic testing technique employed to synthesize real-world corner cases, which can simulate unexpected changes in the working conditions of a DNN-based system. In particular, we will introduce the image transformations and search strategy exploited.

1) Image Transformations: There is a growing body of research on DNN testing that applies metamorphic testing techniques to enrich their test cases [12], [13], [23], [57], [58], [67], [77]. They find that classical image transformations, like rotation, are capable of effectively mimicking changing working environment and exposing erroneous behaviors of DNNs in practice [13], [23], [57], [58], [67]. Note that as one cannot cover all types of corner cases, we employ well-recognized transformations (brightness adjustment, contrast adjustment, rotation, shear, scale, and translation) to synthesize corner cases following [67]. Compared to other transformations, our preliminary study also confirms that selected ones can both generate effective real-world corner cases and reserve original labels of images through controllable parameters, avoiding introducing problematic test cases.

Image brightness is a measure of color intensity. Therefore, to change the brightness of an image, we can increase or reduce all the current pixel values by a constant bias β. The contrast of an image, as its name implies, is determined by the amount of color and luminance differentiation that exists between various objects in the image. We can manipulate the contrast of an image through multiplying all the current pixel values by a constant gain α. The brightness and contrast of an image can frequently change in practice due to the variation of illumination. Rotation, shear, scale, and translation compose the whole set of affine transformations, which is common geometric deformation that happens to captured images due

TABLE I
TRANSFORMATION MATRICES OF AFFINE TRANSFORMATIONS COVERED IN THIS PAPER.

Affine Transformation	Transformation Matrix	Parameters	Example
Rotation	$\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 1 \\ 0 & 0 & 1 \end{bmatrix}$	θ : the rotation angle	
Shear	$\begin{bmatrix} 1 & s_h & 0 \\ s_v & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	s_h : shear ratio along x axis s_v : shear ratio along y axis	
Scale	$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$	s_x : scale ratio along x axis s_y : scale ratio along y axis	
Translation	$\begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix}$	T_x : shift along x axis T_y : shift along y axis	

to perspective irregularity. We leverage them to simulate the distorted appearance of an object resulting from varying camera positions and the shift of this object, which can frequently occur in the real world as well.

Using homogeneous coordinates can formulate affine transformations succinctly. In homogeneous coordinates, we first extend the original coordinates of a two-dimension image $\mathbf{I} = (a, b)$ into homogeneous coordinates $\mathbf{I} = (a, b, 1)$ with three dimensions. Then the coordinates of an affinely transformed image $\mathbf{I}^{'} = (a^{'}, b^{'}, 1)$ are merely the dot product of the corresponding transformation matrix $\mathbf{T}$ and $\mathbf{I}$ (*i.e.,* $\mathbf{I}^{'} = \mathbf{T} \cdot \mathbf{I}$). We list the transformation matrices of four kinds of affine transformation in Table I.

Complement is a different type of image transformation that flips all the pixel values of an image. For example, in the complement of a binary image, black and white are reversed. For greyscale images, their complements are still clearly distinguishable and familiar to human observers, whereas the complements of color images look peculiar and are unlikely to appear in reality. So we only apply this transformation to greyscale images. Since the model never sees the complement of an image during training, it can be regarded as a kind of corner cases as well.

2) Search Strategy: As introduced before, all the image transformations we covered except for complement can be parameterized with different strength, so the first question is how to determine the most suitable parameters for them. A small degree of deformation is scarcely sufficient to reproduce real-world corner cases and disclose the vulnerability of DNNs, while too much distortion may compromise the semantic meanings of seed images and render them unrecognizable. We resolve this issue by grid search in a trial-and-error fashion.

In short, for the sake of exploring a variety of real-world corner cases, we start by applying single transformations to each seed image in turn. We then follow by performing the combination of different image transformations. To decide the parameter for single transformations, we apply it to a fixed set of clean test images with growing distortion strength iteratively. During the search, we also monitor whether the altered images preserve their original semantic meanings. The search stops when the average accuracy of the model on the transformed image set starts to drop by a notable margin. We take it as the omen that the DNN classifier is unfamiliar with the distorted test images and working in trouble. We hold the resultant synthetic test images for the following experiments.

Combining these transformations also contributes to discovering new corner cases. However, it is computationally inhibitive to explore all combinations exhaustively. Therefore we mainly consider the combination of two transformations. The idea of figuring out the most suitable parameters here is similar to that discussed before. Small modification according to empirical observation is explained in Section IV-B. Although the most suitable parameter choices rely on subjective judgments, we also collect error-inducing samples via transformations with a broader range of parameters. We evaluate Deep Validation under this dynamic setting in Section IV-D6.

B. Deep Validation

Figure 1 outlines the framework of Deep Validation. In a word, we take a trained DNN model and add probes into every layer internal to it. During inference, rather than taking the classifier as a black-box oracle and trusting its final decisions blindly, we validate the internal states of the model, namely, outputs of the layer i through discrepancy estimation d_i. We quantify the validity of an input image by its joint discrepancy. Once the total discrepancy exceeds a preset threshold ϵ, we will mark the test image as an invalid input, namely, corner cases that may lead to unexpected behaviors of the system. In this way, we seek to ascertain that the model functions correctly and there is no sign of tampering.

More specifically, we make sure that the internal states of a running DNN classifier follow consistent patterns observed from the training samples when making the same prediction. Otherwise, it is risky to accept its decisions due to rare training samples similar to the ones it confronts. We at first justify the fundamental idea of Deep Validation that validating internal states of a model assists in alleviating its misbehavior. Then we show how to turn to the training data to quantify the validity of inputs via computing discrepancy, which estimates its distance to the region where the probability density of training data resides.

1) Justification: In a traditional program, complex tasks are usually divided into smaller ones and conquered by individual modules separately. We can hence assign these sub-tasks to different developers. In order to make these modules work together seamlessly, it is a good practice to work out a detailed specification first. The specification of a module generally elaborates on what tasks it is supposed to complete, what are the expected inputs and outputs, and so on. Data validation is a beneficial way to guarantee that every component within a large program abides by the respective specification and functions correctly during collaboration [1], [2], [9], [19], [51], [60].

Similarly, as introduced in Section II, a DNN classifier can also be regarded as a sophisticated program. Its layers are small components that summarize the raw input image into increasingly abstract forms, which are usually called the

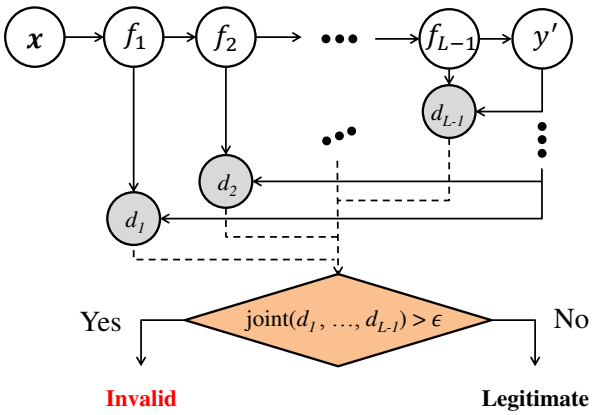


Fig. 1. Deep Validation framework: f_i is short-hand for the output of the i^{th} hidden layer where $i \in \{1, 2, ..., L-1\}$. y' means the predicted label for input $\mathbf{x}$. d_i is the discrepancy estimation for the output of the i^{th} hidden layer.

hidden representations of the image. Recall that each layer can only perform a specific simple computation and owns relatively much-limited capability. It is thus apparent that each layer has an input domain where it has learned to work well. We can regard this region as its valid input range.

Under normal circumstances, since the test images follow the same distribution of the training data, the raw images are mapped into valid input space of each inner layer sequentially, and all layers cooperate in harmony as a whole. Therefore it is enough for images to escape from the "familiar" input region of any middle layer to crack the whole system. Worse still, due to the high dimensionality of input space, small unsafe perturbations can be exacerbated when pushed forward along layers [16], [53]. As a result, the contaminated outputs of internal layers can gradually deviate from the regular input areas of succeeding layers and bring about false predictions in the end. Therefore, validating the inputs/states of all middle layers is conducive to spotting abnormal images that the DNN classifier has not yet learned to handle and preventing the misuse of each component layer.

2) Discrepancy Estimation: In traditional programs, we can develop data validation routines in line with detailed program specification. Nevertheless, the obstacle in the context of DNN models is that the legitimate input range for every layer is ill-defined. It is because the decision functions of these layers are learned on their own rather than manually designed by the developers. Moreover, the classification rules they derive from the training data are encoded in millions of parameters, which are nearly impossible to translate. We instead circumvent this difficulty by backtracking the training data.

Since the model has learned to work well on training samples, their hidden representations are supposed to outline the valid working areas of corresponding layers. These legitimate regions, however, can still be too complicated to depict. We hence further decompose the valid input domain of each layer according to different image classes. Recall that in order to tell different categories of images apart, each layer within a DNN should carefully abandon redundant features and retain discriminative ones until only the label information survives in the end. Consequently, images of different classes can fire different patterns and follow different paths when transferred from one area into another one when going through layers. Based on this observation, Deep Validation proceeds by validating the following claim for each intermediate output of a test image:

Whether the output stays near the region where the corresponding hidden representations of training images with the same label concentrate.

Whenever there are considerable discrepancies, the prediction for the test image is no longer credible, and this abnormal input is likely to be an error-inducing corner case, because some components within the DNN classifier are compelled to extrapolate in unknown regions or the input has gone through a strange mapping route.

As a result, we begin by portraying the regions where training images of different categories locate layer by layer. We then mark them as reference distributions. When a new test image comes, we evaluate its divergence regarding the corresponding reference distributions to determine whether the test image is legitimate.

We build our discrepancy estimation on the method proposed by Schlkopf *et al.* [62] to efficiently model the reference distributions. Roughly speaking, their method aims to locate the separating hyperplane of training points after casting them into kernel space. We leverage this approach to capture the region where the probability density of training images resides. Specifically, for each reference distribution, we train a one-class support vector machine (SVM) on the corresponding set of hidden representations of training images. We follow their idea to learn the decision function through requiring its values to be non-negative on small input region where most of the training data spread while negative otherwise.

After that, we approximate the validity of a given sample by calculating its signed distance to the learned supporting hyperplane in kernel space. By doing so, we circumvent the difficulty in depicting data whose underlying distribution is too elusive to express explicitly. On top of that, since we simplify the reference distribution through a careful segmentation of training samples, it helps the training of one-class SVM to scale well to high-dimension data.

Algorithm 1 elucidates the procedure for training these one-class SVMs. In simple terms, we begin by removing training images misclassified by the model, since they are likely to be outliers and will do harm to the training of SVMs. Then in each layer except for the last one, we get the hidden representations of all the training images and group them based on their original labels. Each subset of these points is applied to fit one SVM, and therefore we conduct SVM training repeatedly for every class in every layer. The training procedure $SVMTRAIN$ is based on the implementation of the algorithm of Schlkopf *et al.* in the scikit-learn library [4], [56]. We denote the final distance function of SVM(i, k),

Algorithm 1 One-class SVM Training

Require: CNN classifier f, layer number L, class number N, and training data set X_{train}

// *obtain correctly classified images from the training dataset*
$X_{train} \leftarrow \{\mathbf{x}^{(t)} \in X_{train} : y^{(t)} == f(\mathbf{x}^{(t)})\}$
for layer $i \in \{1, ..., L-1\}$ **do**
  for class $k \in \{1, ..., N\}$ **do**
    // *select corresponding training data*
    $X^k \leftarrow \{\mathbf{x}^{(t)} \in X_{train} : y^{(t)} == k\}$
    // *get hidden representations in a specific layer*
    $X_i^k \leftarrow \{f_i(\mathbf{x}^{(t)}) : \mathbf{x}^{(t)} \in X^k\}$
    // *train one-class SVM*(i, k)
    SVMTRAIN(X_i^k)
  end for
end for

Algorithm 2 Discrepancy Estimation

Require: CNN classifier f, layer number L, and test image $\mathbf{x}_{test}$

$y^{'} \leftarrow f(\mathbf{x}_{test})$
for layer $i \in \{1, ..., L-1\}$ **do**
  // *compute discrepancy* d_i *for layer* i
  $d_i \leftarrow DISCREPANCY(y^{'}, f_i(\mathbf{x}_{test}))$
end for
// *evaluate joint discrepancy* d *for the test image*
$d \leftarrow joint\ (d_1, d_2, ..., d_{L-1})$

namely, the signed distance of input to its decision hyperplane, as t_i^k. Here SVM(i, k) is the SVM trained with hidden features of images from class k in layer i.

Algorithm 2 describes the routine to estimate discrepancy of a test image $\mathbf{x}_{test}$. In order to obtain the discrepancy estimation for the intermediate output $f_i(\mathbf{x}_{test})$ of the test image in the i^{th} layer, we first obtain its label prediction $y^{'}$ to index the reference SVM $(i, y^{'})$. Next, we feed $f_i(\mathbf{x}_{test})$ to the corresponding SVM distance function and directly define the opposite as the discrepancy value:

$$DISCREPANCY(y^{'}, f_i(\mathbf{x}_{test})) := - t_{y^{'}}^k(f_i(\mathbf{x}_{test})). \quad (2)$$

It is because we want to have positive discrepancy values for outliers and negative ones otherwise. Finally, we formulate the total discrepancy d as the unweighted sum of discrepancy estimations of all layers:

$$d = joint(d_1, d_2, ..., d_{L-1}) := \sum_{i=1}^{L-1} d_i. \quad (3)$$

This simple joint function turns out to work well. Still, we can further explore it since better combination can lead to more precise estimation.

TABLE II
MODEL ARCHITECTURE FOR SVHN.

Layer Type	Parameters
Convolution + ReLU	64 filters (3 × 3)
Convolution + ReLU + Max Pooling(2 × 2)	64 filters (3 × 3)
Convolution + ReLU	128 filters (3 × 3)
Convolution + ReLU + Max Pooling(2 × 2)	128 filters (3 × 3)
Fully Connected + ReLU	256
Fully Connected + ReLU	256
Softmax	10

TABLE III
MODEL ACCURACY ON TEST DATA.

Dataset	Accuracy on Test Data	Mean Top-1 Prediction Confidence
MNIST	0.9943	0.9979
CIFAR-10	0.9484	0.9456
SVHN	0.9223	0.9878

IV. EXPERIMENTS

A. Experimental Setup

We consider three standard datasets for image classification: MNIST [31], CIFAR-10 [27], and SVHN [46]. We note that SVHN is a relatively "noisy" dataset without much data preprocessing effort in advance. It has two formats of which we utilize the one made up of 32-by-32 color images with cropped digits. We engage the standard training-test partitions of all these datasets.

We utilize pre-trained models for MNIST and CIFAR-10 for the sake of fair comparisons in the following experiments. The MNIST model is a seven-layer CNN [74], and the CIFAR-10 model is DenseNet [25], [39], which has 40 layers in total. We train a seven-layer CNN for SVHN, and Table II summarizes its architecture. We adopt an Adadelta optimizer [75] during training, with an initial learning rate of 1.0 and a decay factor of 0.95. We train the model for 60 epochs with a batch size of 128. We do not apply any data augmentation during training. Table III presents the mean accuracy and prediction confidence of these models on test datasets. Their performances are all comparable to the state-of-the-art results [3].

B. Corner Case Generation

We fix a clean seed image set of 200 images for each model. They are randomly sampled from each test dataset respectively. We make sure that all get correctly classified before any modification. These seed images are leveraged to synthesize corner cases according to the strategies introduced in Section III-A. Table IV lists the search range and step size for each transformation. We note that we only alter the seed images from MNIST by complement since the other datasets are all color images.

At each iteration, we evaluate the $accuracy$ of the target classifier on synthetic images and define the success rate of each configuration as $1 - accuracy$. As we expect, different transformations have different destructive power in different datasets. Some transformations degrade the accuracy of target classifiers quickly with increasing distortion, while others fail

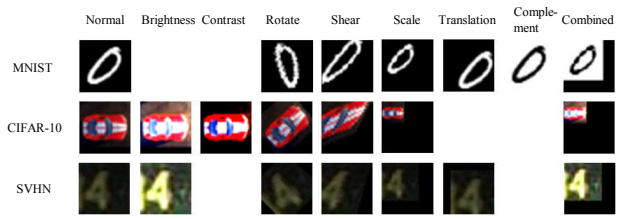


Fig. 2. Examples of synthetic corner cases.

TABLE IV
TRANSFORMATIONS AND SEARCH SPACE UTILIZED WHEN SYNTHESIZING CORNER CASES.

Transformation	Parameter	Parameter Range and Search Step
Brightness	bias β	0 through 0.95, step 0.004
Contrast	gain α	0 through 5.0, step 0.1
Rotation	rotation angle θ	1° through 70°, step 1°
Shear	shear vector (s_h , s_v)	(0, 0) through (0.5, 0.5), step (0.1, 0.1)
Scale	scale vector (s_x , s_y)	(1, 1) through (0.4, 0.4), step (0.1, 0.1)
Translation	translation vector (T_x , T_y)	(0, 0) through (18, 18), step (1, 1)
Complement	maximum pixel value 1.0	-

to convert legitimate samples into error-inducing counterparts until they become hardly discernible.

For individual transformation, the search stops when it obtains a success rate of about 60%. In the following experiments, we do not consider transformations that cannot achieve a success rate of greater than 30% in the end. Combined transformations are mainly meant to enrich the corner cases, and therefore we directly utilize the final parameters above to parametrize component transformations. We select one transformation combination for each dataset that results in the smallest deformation because it can preserve the semantic meaning of images and test the sensitivity of detectors.

Table V lists the success rates of all settings along with the final parameters we employ. Figure 2 illustrates some examples of resultant corner cases. No images are given for transformations with less than 30% success rate. Although our target models obtain exceptional accuracy on clean test data, they are susceptible to these unusual corner cases and undesirably show high confidence in their wrong predictions. It once again reveals the serious threat real-world corner cases can pose.

C. One-class SVM Training

We note that training one-class SVMs merely utilizes the clean training data introduced in Section IV-A. To facilitate the selection of SVM parameters, we leave out 1000 examples as validation data from each training dataset. We apply the same training parameter for all the SVMs within the same layer (*i.e.,* SVM$(i, k) : k \in \{1, ..., N\}$). Nevertheless, they vary from layer to layer since the intermediate outputs from different layers are usually of substantially different dimensions. We note that the overhead of the proposed framework is low, because it is much cheaper to train one-class SVMs than

TABLE V
SUCCESS RATES OF DIFFERENT KINDS OF CORNER CASES.

Dataset	Transformation	Configuration	Success Rate	Mean Top-1 Prediction Confidence
MNIST	Brightness	-	-	-
	Contrast	-	-	-
	Rotation	$\theta = 50°$	0.62	0.8854
	Shear	(s_h , s_v) = (0.2, 0.3)	0.64	0.8491
	Scale	(s_x , s_y) = (0.8, 0.8)	0.665	0.9031
	Translation	(T_x , T_y) = (4, 3)	0.625	0.9118
	Complement	maximum pixel value 1.0	0.53	0.8507
	Combined Transformations	complement combined with scale	0.87	0.8645
CIFAR-10	Brightness	$\beta = 0.51$	0.625	0.7213
	Contrast	$\alpha = 4$	0.585	0.7405
	Rotation	$\theta = 40°$	0.655	0.7782
	Shear	(s_h , s_v) = (0.5, 0.4)	0.59	0.7325
	Scale	(s_x , s_y) = (0.6, 0.6)	0.585	0.4608
	Translation	-	-	-
	Complement	-	-	-
	Combined Transformations	brightness adjustment combined with scale	0.855	0.3891
SVHN	Brightness	$\beta = 0.49$	0.575	0.8315
	Contrast	-	-	-
	Rotation	$\theta = 44°$	0.615	0.9396
	Shear	(s_h , s_v) = (0.3, 0)	0.605	0.9362
	Scale	(s_x , s_y) = (0.7, 0.7)	0.715	0.9524
	Translation	(T_x , T_y) = (5, 5)	0.63	0.9405
	Complement	-	-	-
	Combined Transformations	brightness adjustment combined with scale	0.865	0.9731

training DNNs. As the hidden representations of input images are already available when running DNN systems, querying SVMs also incurs negligible costs. Besides, the training and validation pipeline can be parallelized based on our design.

The target model for CIFAR-10, DenseNet, contains 40 layers. Therefore it takes much more time if we train SVMs for all layers. Thanks to the dense inter-connections between layers, it is convenient for the latter layers to receive the outputs from the former ones directly. Accordingly, errors happen in the early layers can also smoothly propagate to the latter ones, which means that it may be enough to validate the inputs of the rear layers. Based on this observation, Deep Validation only works on the last six layers of DenseNet. However, we envisage validating internal inputs of all layers can make further improvements.

D. Corner Case Detection

1) Evaluation Dataset: As shown in Table V, we have six kinds of successful corner cases for each dataset. Therefore, for each target classifier, there are 1200 synthetic corner cases. We sample the same number of images from the corresponding clean test dataset. They together compose the evaluation dataset. According to whether these corner cases get misclassified or not, we further group them into successful corner cases (SCCs) and failed corner cases (FCCs).

2) Evaluation Metric Definition: The ROC-AUC score is a widely recognized metric to assess an anomaly detection method in similar tasks [6], [14]. It takes both the false positive rate (FPR) and the true positive rate (TPR) into consideration. Since some corner cases fail to fool the target model, how to define true positives is the first problem we need to address. Although detectors should also label failed corner cases as true positives under some application-specific requirements, we follow the practice in adversarial machine learning to put aside failed corner cases first [74]. We defer further discussions

to Section IV-D6. Consequently, the true positive rate means the proportion of detected SCCs in all SCCs present. Because taking clean samples for true positives is undesired in most cases, we define the false positive rate as the percentage of normal instances mislabeled by the detectors.

3) Detection Results: Figure 3 describes the distributions of the normalized discrepancy estimation d in three datasets. As we expected, almost all legitimate images have negative discrepancy values while the opposite holds for SCCs. One can set the center of both distribution centroids as the discrepancy threshold ϵ. It is a reasonable trade-off between achieving high true positive rates and remaining relatively small false positive rates.

We report ROC-AUC scores of Deep Validation in Table VI. We call the whole set of SVMs in the i^{th} layer as the i^{th} single validator. The "Single Validator" row indicates the detection result when we only exploit the discrepancy estimation from the specific single validator. We show the best result for each transformation among these single validators in the "Best Transformation-specific Single Validator" row. It depicts the best results single validators can achieve when they are allowed to adapt to different settings by choosing corresponding top performers. A joint validator represents the actual Deep Validation system we deploy, and the last row of every dataset shows its performance.

For the MNIST model, the best single validators against specific transformations all lie in the first three layers. The first and third single validator each can resist one-half transformations most effectively. Because the target model never sees the synthesized corner cases during training, these corner cases may cause great discrepancy once entering the system, which renders the former validators to be immediately aware of them. However, the last validator possesses the most balanced detection capacity, which makes it stand out in the fight against all corner cases. We suspect the reason is that SCCs are pushed so far away from the normal distribution that during inference, they cannot return to the valid input region of the last layer. Therefore, the last validator gets a chance to spot them.

As for CIFAR-10, now each of the last two validators performs best under one-half transformations respectively. The penultimate validator is the best candidate that can handle all transformations when working alone. It supports our design in Section IV-C to focus on the validation of rear layers. Finally, for single validators in the SVHN model, the best players return to the first two layers except for shear transformation, where the penultimate validator outperforms the others. The accumulation of small discrepancies throughout former layers may explicate why the penultimate validator can observe larger discrepancy values.

Notably, the last single validators in the SVHN model are less capable of distinguishing SCCs from clean images, which leads to deteriorated detection performance of the joint validator, especially for scale. It may imply that some corner cases have been transferred into "safe" regions by former layers mistakenly, where normal samples may concentrate as well. As a result, the last layer thinks that these are the legitimate inputs it has seen before and is confident about its predictions. As shown in Table V, the relatively high confidence of the SVHN model on its wrong predictions corroborates our reasoning.

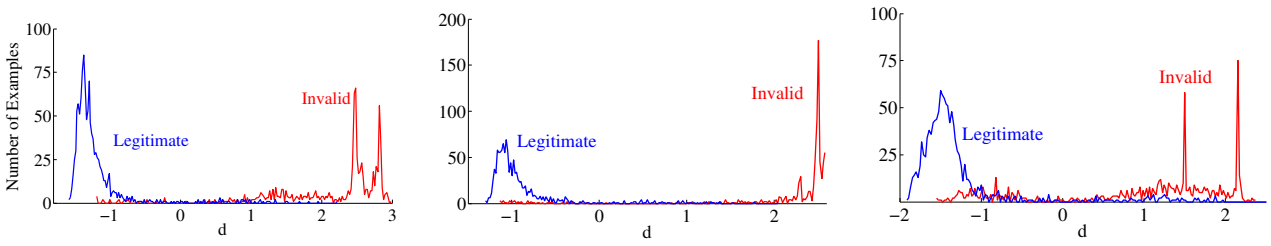
Since different single validators are capable of coping with different transformations, it inspires us to build up a versatile detector through listening to all their opinions. Also, when dealing with invalid inputs, layers are working in unhealthy conditions, which can cause the shift of these samples toward legitimate ones along layers by mistake. As such, the performance of single validators can fluctuate. Therefore, combining them as a joint validator can improve and stabilize the performance.

The fact that joint validators obtain the best ROC-AUC scores under most settings evidences the idea. In MNIST and CIFAR-10, joint validators always outperform single validators except for brightness adjustment in CIFAR-10. It may result from the unsatisfactory performance of the second validator. Nevertheless, this is compensated by the latter layers as the discrepancies can propagate along inter-connections. Therefore the degradation of the joint validator in this configuration is negligible. As for SVHN, the joint validator precedes best single validators in addressing rotated images, while in the other scenarios, it lags slightly behind the best single validators. We suppose the unstable performance of single validators is the main reason. However, it can be improved via carefully assigning different weights to different single validators when computing joint discrepancy values, rather than adopting equal importance here.

We note that joint validators achieve best overall ROC-AUC scores across three datasets, which again corroborates their effectiveness in detecting a variety of invalid inputs (*i.e.,* SCCs). When constraining the overall false positive rate to be around 3%, 7%, and 11% on MNIST, CIFAR-10, and SVHN respectively, joint validators can achieve a respective overall true positive rate (*i.e.,* detection rate) over 96%, 94%, and 90%.

4) Comparison with Adversarial Image Detection Methods: Adversarial images can fool a DNN classifier by attaching imperceptible additive perturbations to clean ones after acquiring white-box access. Methods that succeed to filter out these samples are supposed to capture the intrinsic properties of normal images. In contrast, real-world corner cases seem to introduce much larger distortions. Hence one may think such detection mechanisms can also identify real-world corner cases with ease.

In order to verify this conjecture, we choose two representative detection methods, feature squeezing [74] and kernel density estimation [14], that both report state-of-the-art detection results on a crowd of adversarial images. We directly adopt their original implementations and deploy them according to the descriptions in their work. For feature squeezing, they exploit the same MNIST and CIFAR-10 models we experiment with, so we employ the same squeezer (*i.e.,* detector) configurations as they suggested. Since they do not consider SVHN dataset, we try out the best squeezer combination they



(a) MNIST (b) CIFAR-10 (c) SVHN

Fig. 3. Discrepancy distributions of legitimate images and invalid ones (successful corner cases). Each plot is based on 200 histogram bins and fitted over discrepancy estimations for corresponding evaluation dataset.

TABLE VI
ROC-AUC SCORES OF DEEP VALIDATION.

Dataset	Configuration		Transformation Method Used to Synthesize Corner Cases								Overall ROC-AUC Score
	Validator	Layer No.	Brightness	Contrast	Rotation	Shear	Scale	Translation	Complement	Combined Transformations	
MNIST	Single Validator	1	-	-	0.8760	0.9987	0.8827	0.8952	**1.0000**	**1.0000**	0.9440
		2	-	-	0.9200	0.9719	0.8048	0.8893	**1.0000**	0.9996	0.9324
		3	-	-	0.9741	0.9797	0.9591	0.9728	0.9850	0.9197	0.9618
		4	-	-	0.9740	0.9823	0.9224	0.9657	0.9876	0.9670	0.9657
		5	-	-	0.9732	0.9788	0.9053	0.9602	0.9861	0.9630	0.9601
		6	-	-	0.9659	0.9889	0.9237	0.9620	0.9871	0.9786	0.9676
	Best Transformation-specific Single Validator		-	-	0.9741	0.9987	0.9591	0.9728	**1.0000**	**1.0000**	0.9676
	Joint Validator		-	-	**0.9891**	**0.9991**	**0.9881**	**0.9844**	**1.0000**	**1.0000**	**0.9937**
CIFAR-10	Single Validator	34	0.7615	0.7579	0.8154	0.8828	0.9670	-	-	0.9752	0.8669
		35	0.5119	0.8749	0.9175	0.8765	0.9339	-	-	0.9099	0.8412
		36	0.7111	0.8230	0.8821	0.8557	0.9722	-	-	0.9497	0.8706
		37	0.8958	0.9664	0.9109	0.9315	0.9988	-	-	0.9993	0.9528
		38	**0.9674**	0.9788	0.9272	0.9351	0.9988	-	-	0.9992	0.9692
		39	0.9321	0.9566	0.9245	0.9353	**1.0000**	-	-	**1.0000**	0.9603
	Best Transformation-specific Single Validator		**0.9674**	0.9788	0.9272	0.9353	**1.0000**	-	-	**1.0000**	0.9692
	Joint Validator		0.9550	**0.9882**	**0.9561**	**0.9787**	**1.0000**	-	-	**1.0000**	**0.9805**
SVHN	Single Validator	1	**0.9887**	-	0.8367	0.7200	0.9824	0.9307	-	0.9992	0.9168
		2	0.8880	-	0.9006	0.7988	**0.9934**	**0.9664**	-	**0.9994**	0.9317
		3	0.8186	-	0.8714	0.7385	0.9141	0.8993	-	0.9980	0.8831
		4	0.7696	-	0.8650	0.8596	0.9057	0.8958	-	0.9858	0.8887
		5	0.8923	-	0.8759	**0.8930**	0.9314	0.9124	-	0.9940	0.9220
		6	0.9602	-	0.8631	0.8075	0.7784	0.8737	-	0.9007	0.8633
	Best Transformation-specific Single Validator		**0.9887**	-	0.9006	**0.8930**	**0.9934**	**0.9664**	-	**0.9994**	0.9317
	Joint Validator		0.9638	-	**0.9181**	0.8729	0.9757	0.9510	-	0.9979	**0.9506**

offered. For kernel density estimation, we train and fine-tune their detectors on the same data we leverage.

The comparison between Deep Validation and these two benchmark approaches, however, provides a surprising counter-example as shown in Table VII. Despite their tremendous success against numerous white-box attacks, both detection methods disappoint us in the face of real-world corner cases. Our Deep Validation, on the contrary, overwhelmingly dominates the competition. Kernel density estimation can hardly mitigate real-world corner cases as its ROC-AUC scores are below 0.26 across all three datasets. We suspect the reason is that unlike our method, they rely on only one layer and mix all the clean images from different classes together. It is therefore difficult to precisely depict the complicated distribution.

Feature squeezing surpasses kernel density estimation on all experiments. However, its performance is still consistently inferior to ours. As introduced before, even the clean images in the SVHN dataset are a little noisy, which makes it hard to characterize their distribution. Consequently, it is remarkable that we prevail by a significant margin in the SVHN dataset, and there is enough reason to doubt whether feature squeezing indeed captures the very nature of the normal data. The severe degradation of the detection performance of both benchmark techniques also demonstrates that real-world corner cases may embrace distinct properties, compared to artificially generated adversarial samples. It would be an interesting problem for future study.

5) Use Case in Defending against White-box Attacks: Since our Deep Validation is designed to be oblivious to application scenarios, we expect that it can alleviate white-box attacks as well. We only compare Deep Validation with feature

TABLE VII
COMPARISON WITH FEATURE SQUEEZING AND KERNEL DENSITY ESTIMATION IN DETECTING REAL-WORLD CORNER CASES.

Dataset	Method	Overall ROC-AUC Score (SCCs)
MNIST	Deep Validation	**0.9937**
	Feature Squeezing	0.9784
	Kernel Density Estimation	0.1436
CIFAR-10	Deep Validation	**0.9805**
	Feature Squeezing	0.8796
	Kernel Density Estimation	0.1254
SVHN	Deep Validation	**0.9506**
	Feature Squeezing	0.6870
	Kernel Density Estimation	0.2543

squeezing here because it has been tested under more attacks. We conduct extensive experiments on the MNIST dataset following the same setting as described in [74]. We explore all the white-box attacks that were covered: fast gradient sign method (FGSM) [16], basic iterative method (BIM) [29], Jacobian-based saliency map approach (JSMA) [55], and Carlini/Wagner Attacks (CW_2, CW_∞, and CW_0) [7]. They are all representative and fierce attack methods to date. We utilize the same seed and clean images in the previous evaluation dataset for consistency.

Table VIII enumerates all the results. We adopt the same notations in [74] where "Next" and "LL" mean the next class and least-likely class in reference to the ground truth label respectively. However, successful adversarial samples (SAEs) still mean the ones that cause wrong predictions regardless of their target labels, which is more reasonable from the perspective of defenders. The others are named failed adversarial samples (FAEs), while adversarial samples (AEs) contain both of them.

Overall, Deep Validation obtains impressive results comparable with feature squeezing. When only counting SAEs as true positives, Deep Validation slightly falls behind feature squeezing, but the result reverses when also incorporating FAEs as true positives. As adversarial samples are deliberately synthesized images that seldom happen in practice, failed attempts are an apparent sign of intrusion. It is therefore commendable that Deep Validation can outperform feature squeezing in spotting unsuccessful efforts, which is conducive to alerting people for upcoming attacks. Since we focus this paper on real-world corner cases, we do not conduct similar experiments on other datasets, which, however, are worthwhile for future work. We further note that the prominent performance of Deep Validation observed here is not a guarantee that Deep Validation can be immune to arbitrary attacks, but the promising results confirm that it can be combined with other security methods to make the life of attackers harder.

6) Detection Rate Variations under Increasing Distortions: During the search for crafting real-world corner cases, increasing distortion can lead to more and more error-inducing samples in most cases. A prominent detector is supposed to identify all successful corner cases (SCCs) even the ones with slight perturbation. Besides, it is generally more appreciated that detectors should also pay attention to failed corner cases (FCCs) when there is significant distortion, because it means that the system is working at elevated risk of fatal mistakes. On the other hand, gentle image deformation can frequently happen in practice, and we do not want to care about it as the underlying system can adapt to such changes on its own.

Consequently, we conduct similar experiments as in Section IV-D3 to study how the detection rates of Deep Validation and feature squeezing vary with increasing distortion. We do not consider kernel density estimation here due to its poor performance reported before. Figure 4 shows the results for scale transformation in MNIST. The results for other settings show a similar trend and are thus omitted here.

Deep Validation almost keeps 100% detection rates on SCCs with various scale ratios. The drop in scale ratio two is because there are six SCCs in total, and only one escapes from the detection. Increasing the scale ratio is likely to push normal images further away from the distribution they previously stay, which leads to growing detection rates of Deep Validation on FCCs as well. Notably, the growth is proportional to the success rate of corner cases, which is desirable as stated before. It means that Deep Validation is aware of the imminent danger.

As for feature squeezing, it exhibits irregular violent oscillations and trend of deterioration in detection rates with increasing deformation. Worse still, it hardly approaches satisfactory detection rates on SCCs, even when the degradation of model accuracy has become disastrous. This result further confirms the flaws of feature squeezing in handling real-world corner cases.

V. RELATED WORK

A. Testing of Deep Learning Systems

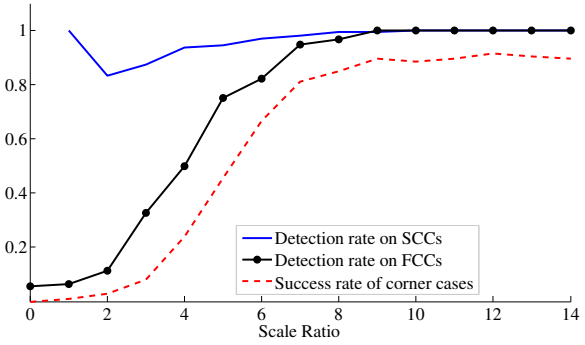
Regular test data often lack diversity and fail to expose stealthy bugs in DNNs. Hence there is a growing interest in automatically generating a mass of test cases, which are capable of simulating different real-world circumstances [57], [58], [67], [77]. In order to bypass the test oracle problem [24], they often resort to differential or metamorphic testing techniques [8], [40]. They generally synthesize test cases by applying image transformation or by taking advantage of Generative Adversarial Networks (GANs) [33]. Such efforts are conducive to enhancing the dependability and safety of DNNs before deployment, but in a running DNN-based system, we still need to monitor the status of the system in case of any unexpected situation it cannot handle. Our work indeed fills this gap.

B. Data Validation

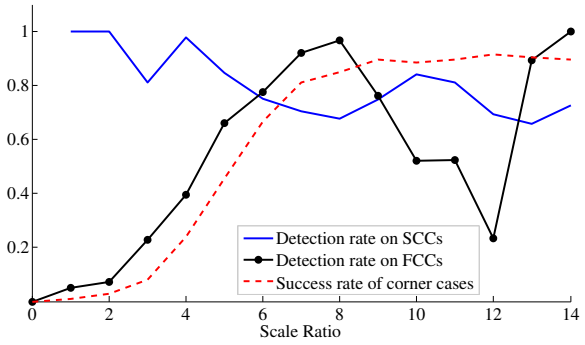
Data validation seeks to protect the system by disallowing the entry of data that violate predefined validation rules [1], [9], [19], [60]. For example, web applications often turn to data validation to prevent input attacks like buffer overflow, SQL injection, and cross-site scripting [2], [51]. Concurrent with our work, Zhang *et al.* propose to validate the inputs of DNNs via VGGNet features [77]. However, they only demonstrate the viability of their method to differentiate images under different

TABLE VIII
COMPARISON WITH FEATURE SQUEEZING IN THE FACE OF WHITE-BOX ATTACKS.

Attack Method		FGSM	BIM	CW_∞		CW_2		CW_0		JSMA		Overall ROC-AUC Score
Target Label		Untargeted	Untargeted	Next	LL	Next	LL	Next	LL	Next	LL	
Success Rate		0.4300	0.9100	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.6650	0.5150	
SAEs	Deep Validation	1.0000	1.0000	0.9992	0.9965	0.9347	0.9758	0.9329	0.9651	0.9851	0.9944	0.9755
	Feature Squeezing	0.9970	0.9972	1.0000	1.0000	0.9993	0.9996	0.9920	0.9920	0.9973	0.9972	**0.9971**
AEs	Deep Validation	1.0000	1.0000	0.9992	0.9965	0.9347	0.9758	0.9329	0.9651	0.9282	0.8399	**0.9572**
	Feature Squeezing	0.9441	0.9691	1.0000	1.0000	0.9993	0.9996	0.9920	0.9920	0.8169	0.6870	0.9400



(a) Deep Validation



(b) Feature Squeezing

Fig. 4. Detection rate with regard to increasing scale ratios in MNIST. Both methods have the same false positive rate of 0.059 on clean data.

weather conditions. Furthermore, they do not investigate the efficacy of their approach in mitigating model misbehavior. So their technique is merely aware of changing weather, no matter whether misclassification occurs, which generally has comparatively limited use in practice. We investigate general misbehavior of DNN classifiers and propose an effective model validation mechanism to enable fail-safe operations in practice.

VI. CONCLUSION

The blind spots of DNN-based systems are more stealthy and challenging to fix compared to traditional programs, rendering them error-prone under real-world corner cases. Current efforts mostly concentrate on developing precaution strategies and neglect the importance of the fail-safe mechanism. Our Deep Validation is the first promising solution to automatically monitor and validate the intermediate inputs/states of running DNN classifiers. It can identify error-inducing inputs and actively call for human intervention when the system is perceived working incorrectly.

Extensive investigations of Deep Validation exhibit its exceptional performance in addressing a broad spectrum of real-world corner cases and sensitivity to approaching risks. The comparisons with well-recognized detection methods of adversarial images further showcase its superiority and promise of complementing existing approaches to make DNN-based systems more safe and dependable.

The limitation of Deep Validation stems from the computation overhead of validating all components. Fortunately, it can be mitigated by ad hoc modifications according to network structures, as we do for DenseNet before. How to offer the flexibility that allows a trade-off between ultra dependability and high efficiency is an exciting direction for future work.

ACKNOWLEDGMENT

The authors are grateful to their shepherd Saman Zonouz and anonymous reviewers for the detailed and valuable comments. This work is supported by the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CUHK 14208815 and No. CUHK 14210717 of the General Research Fund). Hui Xu is the corresponding author.

REFERENCES

[1] V. R. Basili and D. M. Weiss, "A methodology for collecting valid software engineering data." Naval Research Lab Washington DC, Tech. Rep., 1983.

[2] K. Beaver, *Hacking for dummies*. John Wiley & Sons, 2007.

[3] R. Benenson. Classification datasets results. Accessed: July 2018. [Online]. Available: http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html#5356484e

[4] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.

[5] M. L. M. L. Bushnell, *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*, ser. Frontiers in electronic testing. Boston ; London: Kluwer Academic, 2000.

[6] N. Carlini and D. Wagner, "Adversarial examples are not easily detected: Bypassing ten detection methods," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. ACM, 2017, pp. 3–14.

[7] ——, "Towards evaluating the robustness of neural networks," in *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 2017, pp. 39–57.

[8] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: a new approach for generating next test cases," Technical Report HKUST-CS98-01, Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong, Tech. Rep., 1998.

[9] M. Curphey, D. Endler, W. Hau, S. Taylor, T. Smith, A. Russell, G. McKenna, R. Parke, K. McLaughlin, N. Tranter *et al.*, "A guide to building secure web applications," *The Open Web Application Security Project*, vol. 1, no. 1, 2002.

[10] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[11] F. F. dos Santos, L. Draghetti, L. Weigel, L. Carro, P. Navaux, and P. Rech, "Evaluation and mitigation of soft-errors in neural network-based object detection in three gpu architectures," in *Dependable Systems and Networks Workshop (DSN-W), 2017 47th Annual IEEE/IFIP International Conference on*. IEEE, 2017, pp. 169–176.

[12] A. Dwarakanath, M. Ahuja, S. Sikand, R. M. Rao, R. Bose, N. Dubash, and S. Podder, "Identifying implementation bugs in machine learning based image classifiers using metamorphic testing," in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, 2018, pp. 118–128.

[13] L. Engstrom, D. Tsipras, L. Schmidt, and A. Madry, "A rotation and a translation suffice: Fooling cnns with simple transformations," in *NIPS Machine Learning and Computer Security Workshop*, 2018.

[14] R. Feinman, R. R. Curtin, S. Shintre, and A. B. Gardner, "Detecting adversarial samples from artifacts," *arXiv preprint arXiv:1703.00410*, 2017.

[15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[16] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *CoRR*, vol. abs/1412.6572, 2014.

[17] H. Hassan, A. Aue, C. Chen, V. Chowdhary, J. Clark, C. Federmann, X. Huang, M. Junczys-Dowmunt, W. Lewis, M. Li *et al.*, "Achieving human parity on automatic chinese to english news translation," *arXiv preprint arXiv:1803.05567*, 2018.

[18] 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.

[19] K. L. Heninger, "Specifying software requirements for complex systems: New techniques and their application," *IEEE Transactions on Software Engineering*, no. 1, pp. 2–13, 1980.

[20] K. M. Hermann, T. Kocisky, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom, "Teaching machines to read and comprehend," in *Advances in Neural Information Processing Systems*, 2015, pp. 1693–1701.

[21] G. E. Hinton, "Learning multiple layers of representation," *Trends in cognitive sciences*, vol. 11, no. 10, pp. 428–434, 2007.

[22] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[23] H. Hosseini, B. Xiao, M. Jaiswal, and R. Poovendran, "On the limitation of convolutional neural networks in recognizing negative images," in *Machine Learning and Applications (ICMLA), 2017 16th IEEE International Conference on*. IEEE, 2017, pp. 352–358.

[24] W. E. Howden, "Theoretical and empirical studies of program testing," in *Proceedings of the 3rd international conference on Software engineering*. IEEE Press, 1978, pp. 305–311.

[25] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, vol. 1, no. 2, 2017, p. 3.

[26] V. B. Krishna, M. Rausch, B. E. Ujcich, I. Gupta, and W. H. Sanders, "Remax: Reachability-maximizing p2p detection of erroneous readings in wireless sensor networks," in *Dependable Systems and Networks (DSN), 2017 47th Annual IEEE/IFIP International Conference on*. IEEE, 2017, pp. 321–332.

[27] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[28] 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.

[29] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[30] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[31] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, vol. 2, 2010.

[32] K. Lee, K. Lee, H. Lee, and J. Shin, "A simple unified framework for detecting out-of-distribution samples and adversarial attacks," in *Advances in Neural Information Processing Systems*, 2018, pp. 7165–7175.

[33] M.-Y. Liu, T. Breuel, and J. Kautz, "Unsupervised image-to-image translation networks," in *Advances in Neural Information Processing Systems*, 2017, pp. 700–708.

[34] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song, "Sphereface: Deep hypersphere embedding for face recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, 2017, p. 1.

[35] S.-C. Lo, S.-L. Lou, J.-S. Lin, M. T. Freedman, M. V. Chien, and S. K. Mun, "Artificial convolution neural network techniques and applications for lung nodule detection," *IEEE Transactions on Medical Imaging*, vol. 14, no. 4, pp. 711–718, 1995.

[36] J. Lu, T. Issaranon, and D. Forsyth, "Safetynet: Detecting and rejecting adversarial examples robustly," in *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2017, pp. 446–454.

[37] X. Ma, B. Li, Y. Wang, S. M. Erfani, S. Wijewickrema, G. Schoenebeck, M. E. Houle, D. Song, and J. Bailey, "Characterizing adversarial subspaces using local intrinsic dimensionality," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=B1gJ1L2aW

[38] 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*, 2018. [Online]. Available: https://openreview.net/forum?id=rJzIBfZAb

[39] S. Majumdar. (2017) Densenet. Accessed: March 2018. [Online]. Available: https://github.com/titu1994/DenseNet/

[40] W. M. McKeeman, "Differential testing for software," *Digital Technical Journal*, vol. 10, no. 1, pp. 100–107, 1998.

[41] D. Meng and H. Chen, "Magnet: a two-pronged defense against adversarial examples," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 135–147.

[42] J. H. Metzen, T. Genewein, V. Fischer, and B. Bischoff, "On detecting adversarial perturbations," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[43] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *International Conference on Learning Representations (ICLR) Workshop Track*, 2013.

[44] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017, pp. 86–94.

[45] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2574–2582.

[46] Y. Netzer, T. Wang, 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*, vol. 2011, no. 2, 2011, p. 5.

[47] A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and J. Clune, "Synthesizing the preferred inputs for neurons in neural networks via deep generator networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 3387–3395.

[48] A. Nguyen, J. Yosinski, and J. Clune, "Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks," *Visualization for Deep Learning workshop, International Conference in Machine Learning*, 2016.

[49] B. Nie, J. Xue, S. Gupta, T. Patel, C. Engelmann, E. Smirni, and D. Tiwari, "Machine learning models for gpu error prediction in a large scale hpc system," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2018, pp. 95–106.

[50] M. A. Nielsen, *Neural networks and deep learning*. Determination Press, 2015.

[51] OWASP Foundation. (2013) Open web application security project: Data validation. Accessed: September 2018. [Online]. Available: https://www.owasp.org/index.php/Data_Validation

[52] T. Pang, C. Du, Y. Dong, and J. Zhu, "Towards robust detection of adversarial examples," in *Advances in Neural Information Processing Systems*, 2018, pp. 4580–4590.
[53] N. Papernot and P. McDaniel, "Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning," *arXiv preprint arXiv:1803.04765*, 2018.
[54] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ACM, 2017, pp. 506–519.
[55] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE, 2016, pp. 372–387.
[56] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
[57] 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.
[58] ——, "Towards practical verification of machine learning: The case of computer vision systems," *arXiv preprint arXiv:1712.01785*, 2017.
[59] C. Pham, D. Chen, Z. Kalbarczyk, and R. K. Iyer, "Cloudval: A framework for validation of virtualization environment in cloud infrastructure," in *Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*. IEEE, 2011, pp. 189–196.
[60] L. L. Pipino, Y. W. Lee, and R. Y. Wang, "Data quality assessment," *Communications of the ACM*, vol. 45, no. 4, pp. 211–218, 2002.
[61] S. Sankaranarayanan, A. Alavi, C. D. Castillo, and R. Chellappa, "Triplet probabilistic embedding for face verification and clustering," in *The International Conference on Biometrics Theory, Applications and Systems (BTAS)*. IEEE, 2016, pp. 1–8.
[62] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural computation*, vol. 13, no. 7, pp. 1443–1471, 2001.
[63] J. Stewart. (2018, March) Tesla's autopilot was involved in another deadly car crash. [Online]. Available: https://www.wired.com/story/tesla-autopilot-self-driving-crash-california/
[64] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.
[65] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.
[66] Y. Tian, P. Luo, X. Wang, and X. Tang, "Pedestrian detection aided by deep learning semantic tasks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 5079–5087.
[67] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th International Conference on Software Engineering*. ACM, 2018, pp. 303–314.
[68] T.S. (2018, May) Why ubers self-driving car killed a pedestrian. [Online]. Available: https://www.economist.com/the-economist-explains/2018/05/29/why-ubers-self-driving-car-killed-a-pedestrian
[69] V. N. Vapnik, *Statistical learning theory*, ser. Adaptive and learning systems for signal processing, communications, and control. New York: John Wiley & Sons, 1998.
[70] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.
[71] J. Wang, H. Ding, F. A. Bidgoli, B. Zhou, C. Iribarren, S. Molloi, and P. Baldi, "Detecting cardiovascular disease from mammograms with deep learning." *IEEE Trans. Med. Imaging*, vol. 36, no. 5, pp. 1172–1181, 2017.
[72] D. H. Wolpert, "The lack of a priori distinctions between learning algorithms," *Neural computation*, vol. 8, no. 7, pp. 1341–1390, 1996.
[73] H. Xu, Y. Gao, F. Yu, and T. Darrell, "End-to-end learning of driving models from large-scale video datasets," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2174–2182.
[74] W. Xu, D. Evans, and Y. Qi, "Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks," in *Proceedings of the 2018 Network and Distributed Systems Security Symposium (NDSS)*, 2018.
[75] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.
[76] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.
[77] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 2018, pp. 132–142.
[78] Z. Zheng and P. Hong, "Robust detection of adversarial attacks by modeling the intrinsic properties of deep neural networks," in *Advances in Neural Information Processing Systems*, 2018, pp. 7923–7932.