Yuehuan Wang, Zenan Li, Jingwei Xu *et al.* Journal of computer science and technology: Predicted Robustness as QoS for Deep Neural Network Models. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 33(1): 1–20 January 2018. DOI 10.1007/s11390-015-0000-0

Predicted Robustness as QoS for Deep Neural Network Models

Yuehuan Wang[1], Zenan Li[1], Jingwei Xu[1,*], Ping Yu[1], Taolue Chen[2,1] and Xiaoxing Ma[1]

[1]*State Key Lab of Novel Software Technology, Nanjing University, Nanjing 210023, China*
[2]*Department of Computer Science, University of Surrey, Guilford, UK*

E-mail: {wangyuehuan, lizenan}@smail.nju.edu.cn; {jingweix, yuping}@nju.edu.cn; taolue.chen@surrey.ac.uk; xxm@nju.edu.cn

Received March 31, 2020; revised July 19, 2020.

Abstract The adoption of Deep Neural Network (DNN) models as integral parts of real-world software systems necessitate explicit consideration of their quality-of-service (QoS). A concerning issue is that DNN models are prone to adversarial attacks, and thus it is vitally important to be aware how robust a model's prediction is for a given input instance. A fragile prediction, even with high confidence, is not trustworthy in light of the possibility of adversarial attacks. We propose that DNN models should produce a robustness value as an additional QoS indicator, along with the confidence value, for each prediction they make. Existing approaches for robustness computation are based on adversarial searching, which are usually too expensive to be excised in real time. In this paper, we propose to predict, rather than compute, the robustness measure for each input instance. Specifically, our approach inspects the output of the neurons of the target model and trains another DNN model to predict the robustness. We focus on Convolutional Neural Network (CNN) models in the current research. Experiments show that our approach is accurate, with only 10%-34% additional errors compared to the offline heavy-weight robustness analysis. It also significantly outperforms some alternative methods. We further validate the effectiveness of the approach when it is applied to detect adversarial attacks and out-of-distribution input. Our approach demonstrates a better performance than, or at least is comparable to, the state-of-the-art techniques.

Keywords Deep Neural Networks, Quality of Service, Robustness, Prediction

1 Introduction

Deep learning (DL) [16] has been demonstrated surprising power in various challenging tasks such as natural language processing [1, 37], speech recognition [8, 39], image processing [7, 14, 34], recommendation systems [52, 53], gaming [31] and even in the sentiment analysis for human beings [54], which are hard to accomplish using conventional methods. Consequently, Deep Neural Network (DNN) models are increasingly adopted in real-world applications, including some safety-critical scenarios such as self-driving [3], disease diagnosis [5, 19], and malware detection [40].

However, different from conventional software artifacts, DNN models provide little guarantee about their Quality of Service (QoS) on each individual input other than the often inaccurate confidence value [57,58]. This is largely due to the inductive nature of statistical machine learning and the lack of interpretability for DNN

Regular Paper

This work was supported by the National Basic Research 973 Program of China under Grant No. 2015CB352202, the National Natural Science Foundation of China under Grant Nos. 61690204, 61802170, and 61872340, the Guangdong Science and Technology Department grant under Grant No. 2018B010107004, the Natural Science Foundation of Guangdong Province of China under Grant No. 2019A1515011689, and the Overseas Grant of the State Key Laboratory of Novel Software Technology under Grant No. KFKT2018A16.

*Corresponding Author

models [59, 62]. Note that statistical metrics such as accuracy, MSE, and F1-measure actually report the model's performance on the testing data, but not how well it works on a new, unseen input.

A particularly naughty problem is that DNNs can be easily fooled by adversarial examples, which are constructed by introducing well-designed human imperceptible perturbations to legitimate examples [6, 22, 24, 25, 35]. The vulnerability to adversarial attacks is prevalent for DNN models, and currently there is no general method to eliminate them despite a plethora of proposals [50, 63]. The *robustness* of a DNN model, which quantifies the model's resilience to adversarial attacks, has attracted a lot of attentions from both machine learning and software engineering communities [11, 60]. Nevertheless, the existing work merely aims at improving the training of models and carries out *offline* robustness analysis, rendering it unsuitable as an instant QoS indication for the model's prediction on the current input. The study of adversarial attacks and robustness of DNN models is somewhat predominately targeted at CNN models. In this paper, for a better comparison with other methods, we focus on CNN models for image classification tasks as well.

In contrast, in this paper, we propose *online* calculation of the robustness for each input at runtime. Whenever a DNN model makes a prediction for an input, a robustness value is quickly estimated to indicate how stable the prediction is, against perturbations over this input. This quality metric is important for the system and user to decide how much the prediction can be trusted, considering the possible perturbation introduced intentionally or unintentionally. Note that the robustness metric is not subsumed by prediction confidence because of the existence of high-confidence adversarial examples [6]. Robustness estimation is also different from outlier detection [60, 61]. The former focuses on the stableness of the current model's computation on the current input, while the latter measures the rareness of an input compared with the whole dataset. Nevertheless, as to be shown in Section 4, we can exploit robustness to detect outliers effectively.

The challenge of online robustness estimation is that it needs to be very lightweight to exercise at runtime. While DNN models are usually trained with powerful computers, they are often deployed in environments with very limited computing power such as mobile and embedded devices. For example, TensorFlow Lite and PyTorch Mobile support DNN model execution on mobile devices [44, 45], but not model training. And while the training process can take hours, and even days or weeks, a trained model gives instant prediction on inputs. The efficiency of robustness calculation needs to match the instant model prediction.

Existing approaches to the robustness analysis are either based on formal verification or adversarial searching. Both of them are too heavy-weight for our purpose. Formal verification approaches take the DNN model as a usual (loop-free) program and try to prove that the output of the model will not change if the perturbation to input is within a small bound. These approaches are computationally expensive owing to the nonlinear structure of DNNs and the high dimensionality of input data. Katz et al. [11] have shown that the problem of verifying robustness for ReLU networks is NP-complete. Despite interesting proposals of leveraging advanced SMT solvers [11, 36] and abstract interpretation [33], currently DNN robustness verification can handle only simple DNN models with very limited perturbation bounds, which makes them impractical for real world applications.

Adversarial searching approaches apply optimization to synthesize adversarial attacks with as small perturbation as possible. The minimum perturbation realizing attack found within a time budget provides a metric for the robustness of the DNN model [4]. Although these approaches scale up to large DNN models, their computation costs are still prohibitively high when used at runtime (more than ten hours for C&W applied on ResNet-50).

We propose to *learn* a robustness predictor for each trained DNN model *offline*. At runtime, for each input fed to the model, one may use the robustness predictor to *predict* how robustness the model is on this input *online*. The insight behind this approach is two-fold. First, robustness is a property about the target DNN model's behavior, which can be observed from the outputs of neurons in the model. Second, the theory of representation learning [59] suggests that deep layers of the target model encode its perception of the input, and thus are informative for the model's robustness on the input. So we build an additional DNN model taking the penultimate layer of the target model as input, and train it with robustness values for target model's training data computed with adversarial searching. Though we focus on CNN models, the underlying principle of our approaches can be generalized to other settings as well.

Extensive experimental evaluations confirmed the efficacy and efficiency of our approach. We first trained eight target DNN models with different architectures (LeNets, VGGs, and ResNet) on different datasets (MNIST, SVHN, and CIFAR-10). For each of them we trained a DNN model as the robustness predictor. Compared with the results from heavy-weight offline robustness analysis, these predictors only introduced 10%-30% additional errors.

Furthermore, robustness predictors were successfully used to detect adversarial examples. The idea is straightforward: DNN models are believed to be unstable on adversarial examples, which usually have very low robustness measure, and thus can be detected by the robustness predictor. It turned out that this method performed better than or at least was similar to other state-of-the art methods based on Kernel Density (KD) [41] and Local Intrinsic Dimensionality (LID) [42].

In summary, the contributions of this paper include:

- The proposal of online prediction of robustness as a QoS measure for DNN models;
- A deep learning based approach to the implementation of the online robustness predictor;
- Effective adversarial example detection based on online robustness predictors; and
- Extensive empirical evaluation confirming the efficacy and efficiency of our approaches.

The idea of predicting robustness for DNN models with DNN models was originally presented in the Internetware conference [66]. In this paper, we substantially extend the work and thoroughly re-write the paper. Especially, we make more comprehensive evaluation of the proposed approach and successfully use the same framework of robustness predictors to detect adversarial example. Moreover, we show the proposed robustness predictor is successfully deployed on the mobile platform with impressive performance, whereas other related methods are incompatible to the existing deep learning frameworks for mobile platforms.

The rest of this paper is organized as follows. We first review the background and related work in Section 2. Section 3 describes the basic idea and the detailed design of the predictor and the detector. Section 4 reports the empirical evaluation of our approach. We conclude our work and discuss future work in Section 5.

2 Background

In this section, we introduce architectures of DNNs, commonly used adversarial attacks and verification for DNN robustness.

2.1 Deep Neural Network

DNN essentially defines a new data-driven programming method, in which the logic is portrayed by a large dataset and layer-wise structures, akin to human brain. DNN models usually are trained by the training dataset with the backpropagation algorithm [30]. In a nutshell, a DNN is built up by the input layer, the hidden layers and the output layer, as shown in Fig. 1. Each hidden layer consists of numerous neurons, which connect to the neurons in the next layer by applying activation functions on the linear weighting with possible bias. The number of neurons in the input layer is equal to the dimension of the input data whereas the output layer generates the probability of every class to which the input instance belongs. In this paper we mainly restrict ourselves to classifiers. However, in general, DNNs could be regarded a function f that transforms a given input to the output.

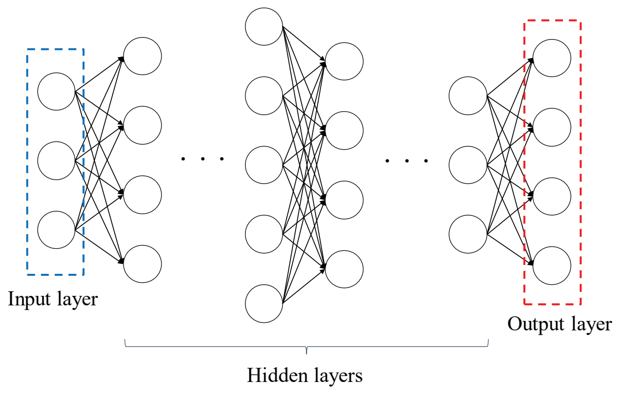


Fig. 1. A simple deep neural network.

2.2 Adversarial Attack

Adversarial examples are derived from natural examples with a crafted perturbation. The difference between the natural example and the adversarial counterpart is usually negligible and thus human imperceptible. These adversarial examples can be generated by numerous adversarial attack methods in a relatively simple way, which can be used to deceive DNNs to make the wrong prediction, revealing serious vulnerability inside DNN models.

In literature, adversarial attacks can be divided into targeted attacks and non-targeted attacks, based on the objective of deception. Given a DNN modeled by the function f with $\boldsymbol{x}$ being the original natural example, adversarial attack techniques aim to craft perturbation $\boldsymbol{\delta}$ such that

- Targeted attack:

$$f(\boldsymbol{x}+\boldsymbol{\delta}) = y_t \tag{1}$$

- Non-targeted attack:

$$f(\boldsymbol{x}+\boldsymbol{\delta}) \neq y_{\boldsymbol{x}} \tag{2}$$

Namely, the objective of the non-targeted attack is to mislead the DNN to predict a label which is different from the ground-truth label $y_{\boldsymbol{x}}$ of $\boldsymbol{x}$. For the targeted attack, the DNN model is guided to generate the given target result y_t.

Some standard adversarial attack techniques in literature are given in order as follows.

- **FGSM & BIM**. Fast Gradient Sign Method (FGSM) [6] is based on the insight that one could find the proper perturbation with the gradient of the cost function. The Basic Iterative Method (BIM) [15] is an iterative variant of FGSM, which adds perturbation to the original input iteratively until the predicted label of the input instance changes. More concretely, the adversarial example is generated according to the following process for FGSM:

$$\boldsymbol{x}^{adv} = \boldsymbol{x}_0^{adv} + \alpha \cdot \text{sign}\left(\nabla_{\boldsymbol{x}} J(\boldsymbol{x}_0^{adv}, y_{true})\right), \quad (3)$$

and the process of BIM could be represented by

$$\boldsymbol{x}_{i+1}^{adv} = \text{Clip}_{\boldsymbol{x},\epsilon}\left(\boldsymbol{x}_i^{adv} + \alpha \cdot \text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{x}_i^{adv}, y_{true}))\right) \quad (4)$$

In both cases, $\boldsymbol{x}_0^{adv}$ is the original input, J is the cost function that is used for training the DNN model, *sign* is the sign function. In particular, $\boldsymbol{x}_i^{adv}$ is the adversarial example generated by the i-th iteration, and *Clip* can be found in [15].

- **C&W**. C&W [4] optimizes the joint objective that minimizes the perturbation and maximizes the probability of fooling DNN simultaneously to generate adversarial example. It utilizes a set of attacking generation algorithms based on different distance metrics including L_1, L_2, and L_∞ norms. Take the L_2-norm based algorithm as an example:

$$\min \quad \|\boldsymbol{\delta}\|_2 + c \cdot f(\boldsymbol{x} + \boldsymbol{\delta}) \quad (5)$$

In particular, by the "change of variables" method, the objective function can be formulated as:

$$\begin{aligned} &\|\frac{1}{2}(\tanh(\boldsymbol{w}) + 1) - \boldsymbol{x}\|_2^2 \\ &+ c \cdot f(\frac{1}{2}(\tanh(\boldsymbol{w}) + 1)) \end{aligned} \quad (6)$$

where $\boldsymbol{w}$ are the newly introduced variables (which can be used to recover $\boldsymbol{\delta}$) and c is a hyperparameter. The first term of the objective is to control the distance between the generated example and the original input, whilst the second term captures the loss function. The details of the C&W algorithm are given in [4].

- **Deepfool**. Deepfool [24] algorithm generates the perturbation by computing the distance from the input to the decision boundary. For the multiclass classification task with a linear classifier $f(\boldsymbol{x}) = \boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{b}$, the minimal perturbation can be generated as follows.

$$\begin{aligned} \arg\min_{\boldsymbol{\delta}} \quad & \|\boldsymbol{\delta}\|_2 \\ \text{s.t.} \quad & \exists c : \boldsymbol{w}_c^\top (\boldsymbol{x} + \boldsymbol{\delta}) + \boldsymbol{b}_c \\ & \geq \boldsymbol{w}_{\bar{c}}^\top (\boldsymbol{x} + \boldsymbol{\delta}) + \boldsymbol{b}_{\bar{c}} \end{aligned} \quad (7)$$

where $\bar{c}$ is a class other than the class c. The details of generalizing the linear classifier to nonlinear ones such as DNNs can be found in [23].

- **JSMA**: Jacobian Saliency Map Attack [43] uses Jacobian matrices to compute the saliency map, which indicates the importance of each dimension in the input with respect to the output. This attack can change the classification result by only modifying a small portion of the input instance. In general, the saliency map can be calculated as follows:

$$S(\boldsymbol{x}, t)[i] = \begin{cases} -\frac{\partial f_t(\boldsymbol{x})}{\partial \boldsymbol{x}_i} \sum_{j \neq t} \frac{\partial f_j(\boldsymbol{x})}{\partial \boldsymbol{x}_i}, \\ \quad \text{if } \frac{\partial f_t(\boldsymbol{x})}{\partial \boldsymbol{x}_i} > 0 \text{ and } \sum_{j \neq t} \frac{\partial f_j(\boldsymbol{x})}{\partial \boldsymbol{x}_i} < 0 \\ 0, \text{otherwise} \end{cases} \quad (8)$$

where $\frac{\partial f_j(\boldsymbol{x})}{\partial \boldsymbol{x}_i}$ is the derivative of the j-th value in the output of DNN model f to the i-th value in the input.

Among the above adversarial attacks, BIM, Deepfool, and C&W aim to generate the minimum perturbation, which can be used to measure robustness for DNNs. As for the concept of **DNN Robustness**, we consider a commonly adopted definition based on the L_2-norm, defined as follows.

$$\begin{aligned} \max \quad & \|\boldsymbol{\delta}\|_2 \\ \text{s.t.} \quad & f(\boldsymbol{x} + \boldsymbol{\delta}) = f(\boldsymbol{x}) \end{aligned} \quad (9)$$

If a perturbed input is within the ball of the original input in Euclidean distance, DNN produces the same classification result as the original input. In other words, DNN is robust against the $\|\boldsymbol{\delta}\|_2$ perturbation.

2.3 Deep Neural Network Verification and Testing

Verifying the robustness of DNNs is a challenging task, which has received considerable attentions [9, 10, 27–29,33,38]. All the work for verification of the robustness of DNNs suffers from scalability issues, and can be only applied to small to medium sized DNNs. The fundamental problem is that to verify a DNN model with ReLU activation function is NP-complete [11]. Many approximation or simplification approaches have been proposed to reduce the time complexity. For example, Releplex and SDP relaxation linearize the ReLU function and relax the verification problem to a convex optimization problem [11, 64]. However, such relaxation is not tight enough because of the nature of the ReLU function, especially with a high number of layers. On the other hand, some work tries to do the complete verification but to limit the region for verification. For example, DeepCheck introduces a novel technique of symbolic analysis for DNN models, and applies them to verify the one-pixel or two-pixel attack [65]. The result is valuable, but applying it into practice still needs more improvement. Thus, we need an efficient and effective method to evaluate the robustness of DNN models.

Similar to testing traditional software, researchers have proposed a series of coverage-based metrics for testing DNN models, most of which are based on the neuron outputs of the DNN model. The higher coverage usually indicates higher test adequacy, meaning that a test set with a high coverage is more likely to trigger bugs, if there are. Neuron coverage was first proposed in DeepXplore [26]. DeepXplore defined whether a neuron is activated or not through a threshold set by users. The percentage of the activated neurons represents the adequacy of the test set. Neuron coverage is coarse-grained for DNN testing, because the neuron coverage can easily reach 100%. DeepGauge [20], a multi-granularity coverages criteria for DNN models, partitions the neuron activation value to multiple sections, and distinguish the corner-case behaviors from the main behaviors. With the concept of combinatorial testing, DeepCT [21] combined the neuron activation states and provided a series of metrics. Partially inspired by the relation between neuron outputs and test adequacy, we aim to predict the robustness using the neuron outputs.

2.4 Adversarial Example Detection

Detecting adversarial examples is another important issue, which aims to determine whether a given input is an adversarial example of the target model [47–51]. Most detection methods are based on more or less the same insight: Although the adversarial example is very similar to the original example, the distributions of these two kinds of examples are different [49]. Hence, an intuitive idea is to use statistical hypothesis tests to detect adversarial examples [49]. However, there is a serious limitation of these hypothesis tests, i.e., it is only able to detect whether a group of examples are adversarial rather than a single input. An alternative proposal is to use statistical models instead of statistical tests. For example, the auto-encoder and the discriminator of GAN were both adopted to measure the distance between the given input and the original examples [47, 48]. Besides, the method based on kernel density estimation was presented to model outputs of the last hidden layer, and to detect the adversarial example based on the uncertainty [41]. Local Intrinsic Dimensionality (LID) was proposed to assess the space-filling capability of the region surrounding a reference example [42], and the experiments showed that LID characteristics could facilitate to highlight adversarial examples.

These methods are however computationally expensive and can only detect adversarial examples lying far from the manifold of the legitimate inputs. More seriously, adversarial perturbations computed based on optimization are always minimal, so the generated adversarial inputs are usually very closed to the decision boundary, which renders these methods ineffective [50, 51].

3 Robustness Prediction for DNNs

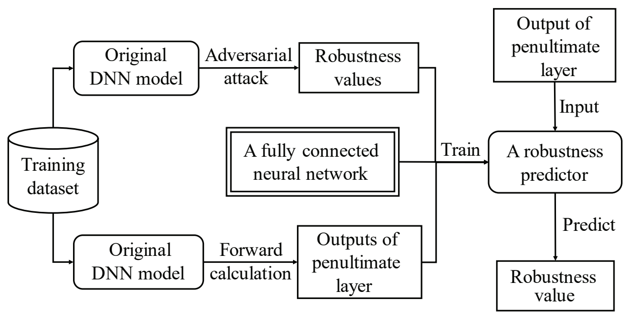


Fig. 2. Overview of the robustness predictor

In this section, we introduce the basic idea of the proposed methods for robustness prediction and adversarial examples detection, which are formulated as *predictor* and *detector* respectively. Fig. 2 illustrates the process of prediction, whereas the process for detection follows the similar structure.

3.1 Rationale

One of the central questions of the current paper is to provide a reliable robustness measure of a given new instance for DNN models. One possible solution is use heavyweight, formal verification based method. However, it is usually very costly and hence infeasible in practice. Alternatively, we propose to use a lightweight method to *predict* such a robustness metric by casting this problem as a regression task, which, interestingly, is to be solved by DNNs. There are, however, certain challenges to overcome, in particular, (1) what features (aka. predictors, covariates, or independent variables) should be used to make the prediction; (2) how to formulate the training set, especially the value of dependent variables in regression.

For the first question, partially inspired by the work on testing NN, we identify neuron output of the DNN on the input instance as the features. For the second question, we build up the training set for prediction based on the train set for the original DNN under consideration. In particular, we exploit adversarial robustness measure, which can be obtained by launching (usually very efficient) adversarial attacks on the DNN. Based on these, we put forward a fast approach to *predict* the robustness of DNN models.

An interesting application of the robustness prediction is to *detect* adversarial examples. A basic observation is that adversarial examples are generated to deceive the DNN model, and are usually less robust than the natural examples. This is especially the case for those which are generated by optimization-based algorithms. To see this, notice that, intuitively, the adversarial perturbation aims to push the input instance to cross over the decision boundary, hence changing the classification result. As optimization-based adversarial attack algorithms craft the minimum perturbation by their formulation, it is reasonable to assume that the generated adversarial examples are close to the decision boundary and thus have fragile robustness. (Cf. Fig. 3 for an illustration of the intuition.) It follows that the robustness measure can be used for distinguishing adversarial examples from natural examples.

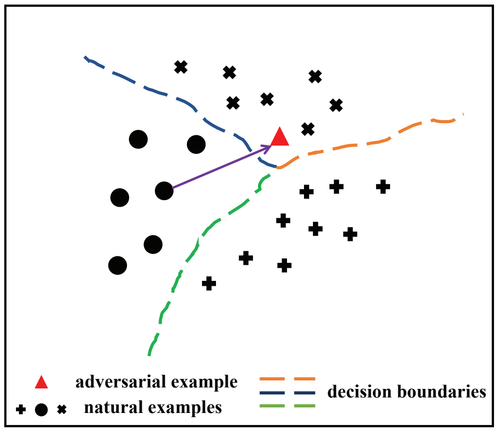


Fig. 3. Robustness for the adversarial example

In the rest of this section, we shall articulate the proposed framework for robustness prediction and adversarial example detection.

3.2 Robustness Measure Prediction

Let M_o be the DNN model under consideration and M_r be the robustness predictor to be trained.

The input to M_r is the neuron output of M_o with respect to an input instance $\boldsymbol{x}$. As the number of all the neurons in M_o may be too high, we only consider the neurons for some specific layers, for instance, the penultimate layer which is generatively considered to encode the features of the input. The neuron output of the layer is obtained through forward computation. Let $\phi_l(\boldsymbol{x})$ be the neuron output vector of the layer l for an input example $\boldsymbol{x}$.

The output of M_r is the robustness measure for the corresponding input, for which we use **attack-based robustness**, i.e., the value returned by optimization-based adversarial attacking algorithms for each input instance. This problem deserves further discussions, as *a priori* it is not clear why these adversarial attacking algorithm provide a sensible estimate of the robustness.

As in Section 2.2, a plethora of adversarial example generation algorithms have been proposed making attacking DNNs a routine task. A commonality of most adversarial attack algorithms is to synthesize the minimal perturbation that can fool the DNN model, We hypothesize that these algorithms perform consistently on this regard, to validate which we design experiments to explore the relation of different measures they generate. In the experiment, we select three adversarial attack algorithms, i.e., C&W, BIM, and Deepfool, to generate the minimum perturbation as the respective robustness measure of the DNN model. We analyze the correlations among the three robustness measures via the Pearson correlation coefficient (PCC) [2], which can illustrate the linear correlation between two sets of data. (Cf. Section 4.1 for details.)

Table 1 shows the experiment results, where we use LeNet-5 and VGG-19 as the DNN models for classifying MNIST and CIFAR-10 datasets, respectively. We can see that the robustness values calculated by different adversarial attack algorithms are highly correlated (in particular, all the PCC values are no less than 0.6). Therefore, it is reasonable to assume that these quantities prescribe consist and sound measures for DNN robustness, which, from a practical point of view, provide a valuable alternative for large-scale DNN models other than costly methods based on formal verification.

The training set for the robustness predictor can then be collected as follows. Recall that the dataset for M_o is composed by pairs of the form $(\boldsymbol{x}_i, y_i)$ for the image example $\boldsymbol{x}_i$ with the classification label y_i. The dataset for the robustness predictor M_r is thus $(\phi_l(\boldsymbol{x}_i), r_i)$, where $\phi_l(\boldsymbol{x}_i)$ is the neuron output vector and r_i is the robustness value.

Table 1. PCC of attack-based robustness measurements

DNN model	Measurement	C&W	Deepfool	BIM
LeNet-5 (MNIST)	C&W	1.0	0.86	0.92
	Deepfool	0.86	1.0	0.85
	BIM	0.92	0.85	1.0
VGG-19 (CIFAR-10)	C&W	1.0	0.73	0.76
	Deepfool	0.73	1.0	0.60
	BIM	0.76	0.60	1.0

For a given example $\boldsymbol{x}_i$, its robustness prediction is given as:

$$r = M_r(\phi_l(\boldsymbol{x})) \tag{10}$$

We note that training the DNN M_r is completed offline and there is no overhead to collect new data online. Moreover, during the training process of M_o, practitioners can calculate the robustness values of all the test examples to assess how robust M_o is.

3.3 Adversarial Example Detection

The detector is designed following the same framework as the robustness predictor. The basic principle is that we hypothesize that natural examples have a higher degree of robustness than adversarial examples. As a result, one can learn a threshold to separate them.

To this end, we use the neuron output from both adversarial examples and natural examples to train the detector model M_d, for which we use the neurons of the penultimate layer as the input of M_d. For a new input of the DNN model, M_d outputs a score to indicate the degree that the input is an adversarial example. For training purposes, the score values of adversarial examples are set to be 0, and 1 for natural examples.

For a given input $\boldsymbol{x}$, the score is computed as follow.

$$s = M_d(\phi_l(\boldsymbol{x})) \tag{11}$$

The lower the score is, the more likely the input is an adversarial example.

4 Evaluation

In this section, we evaluate the performance of our approaches. We focus on the following four research questions:

- **RQ1 (Effectiveness and efficiency of the predictor):** *Can the robustness predictor predict the robustness effectively and efficiently?*
- **RQ2 (Applicability of the predictor):** *Can the robustness predictor be applied to DNN models with different accuracies?*
- **RQ3 (Layer selection):** *Does considering neuron outputs from more layers improve the robustness predictor?*
- **RQ4 (Effectiveness of the detector):** *Can the detector detect adversarial examples effectively?*

4.1 Experiment Setup

Datasets and DNN Models. We select four well-adopted image classification datasets, MNIST [17], CIFAR-10 [13], SVHN [44], and ImageNet [56]. MNIST is a handwritten digit database with 60,000 training examples and 10,000 testing examples, for which we construct three DNN models in the LeNet family [18] (e.g., LeNet-1, LeNet-4 and LeNet-5). CIFAR-10 is an image recognition dataset composed of 50,000 training examples and 10,000 testing examples, for which we construct three DNN models, i.e., VGG-16 with two fully-connected layers [55], VGG-19 [32] and ResNet-50 [7]. SVHN is a house number recognition dataset obtained from Google Street View images containing 73,257 examples for training and 26,032 examples for testing, for which we construct two DNN models, i.e., VGG-16 [32] and ResNet-50 [7]. ImageNet is a large-scale image recognition dataset with more than 1,000,000 images, for which we construct a ResNet-50 [7] model.

Evaluation Metrics. We use the widely adopted Pearson correlation coefficient (PCC) [2] and Mean Absolute Error (MAE) as the metrics for robustness prediction, and Area Under the ROC curve (AUC) for the metric of adversarial example detection.

PCC is a measure of the linear correlation between two variables, which is calculated as follows:

$$\rho_{\boldsymbol{X},\boldsymbol{Y}} = \frac{\operatorname{cov}(\boldsymbol{X},\boldsymbol{Y})}{\sigma_{\boldsymbol{X}}\sigma_{\boldsymbol{Y}}} \tag{12}$$

Table 2. Information about the datasets and DNNs used in our experiments

Dataset	Dataset Description	DNN Model	Neurons of DNN Model	Layers of DNN Model	Accuracy
MNIST	Handwritten digit recognition dataset	LeNet-1	52	7	98.06%
		LeNet-4	148	8	98.43%
		LeNet-5	268	9	96.29%
SVHN	House numbers recognition dataset	VGG-16	14,888	22	94.28%
		ResNet-50	94,059	176	91.71%
CIFAR10	Object recognition dataset	VGG-16	14,888	22	93.59%
		VGG-19	16,168	25	86.44%
		ResNet-50	94,059	176	84.36%

where $\boldsymbol{X}$ and $\boldsymbol{Y}$ are two random variables; $\text{cov}(\boldsymbol{X}, \boldsymbol{Y})$ is the covariance of $\boldsymbol{X}$ and $\boldsymbol{Y}$; $\sigma_{\boldsymbol{X}}$ and $\sigma_{\boldsymbol{Y}}$ are the standard deviations of $\boldsymbol{X}$ and $\boldsymbol{Y}$. The sign of $\rho_{\boldsymbol{X},\boldsymbol{Y}}$ indicates whether $\boldsymbol{X}$ and $\boldsymbol{Y}$ are positively or negatively correlated. (The range of $\rho_{\boldsymbol{X},\boldsymbol{Y}}$ is $[-1, 1]$.) In general, if the absolute value of $\rho_{\boldsymbol{X},\boldsymbol{Y}}$ is more than 0.5, $\boldsymbol{X}$ and $\boldsymbol{Y}$ are considered to be strongly correlated.

MAE measures the absolute value difference between two variables, which is calculated as follows:

$$\text{MAE} = \frac{\sum_{i=1}^{n} |x_i - y_i|}{n} \tag{13}$$

where x_i and y_i are the i-th entry of X and Y with the same dimension n.

AUC is defined as the area under the ROC curve, with the value range $[0.5, 1]$. AUC is based on the True Positive Rate (TPR) and False Positive Rate (FPR) to measure the quality of a classifier. The closer the AUC value is to 1, the better the classifier is for the task.

4.2 (RQ1) Effectiveness and efficiency of the predictor

Each target DNN model is equipped with a robustness predictor whose architectures are shown in Table 3. In our experiments, we distill 10,000 and 1,000 pairs of neuron output and robustness values from the original training and test datasets respectively, which are used to train and test the robustness predictor.

As shown in Table 4, the third column, the robustness values of 1,000 *test* examples predicted by the trained robustness predictor are strongly correlated to the values computed by the C&W method, in particular, all the PCC values are more than 0.7. In Table 5, the value in **Mean of robustness** column is the mean value of 1,000 examples in the test set computed by C&W.

Note that the MAE of VGG-16 is greater than others, this is because, in VGG-16, the images are normalized by subtracting the mean value from each pixel value and then being divided by the standard deviation, while for other datasets, the pixel value is scaled to $[0, 1]$. For the MNIST dataset, each image is 28×28 resolution and the pixel value is scaled to $[0, 1]$. The MAE results show that the robustness predicted by the robustness predictor is sufficiently accurate with 10%-30% errors. These results show that the robustness predictor is effective in robustness prediction.

Table 3. Architecture of the robustness predictors

Dataset	DNN Model	Robustness Predictor
MNIST	LeNet-1	588×120×84×10×1
	LeNet-4	84×120×84×10×1
	LeNet-5	84×120×84×10×1
CIFAR-10	VGG-16	512×256×256×10×1
	VGG-19	512×256×256×10×1
	ResNet-50	2048×256×256×10×1
SVHN	VGG-16	512×256×256×10×1
	ResNet-50	512×256×256×10×1

To further demonstrate the effectiveness of our approach, we consider three measures in literature which

Table 4. PCC for predicting the robustness

Dataset	DNN Model	Robustness Measures			
		Robustness Predictor	Confidence Regression	LSA Regression	DSA Regression
MNIST	LeNet-1	0.92	0.81	0.22	0.61
	LeNet-4	0.95	0.69	0.38	0.68
	LeNet-5	0.95	0.86	0.10	0.08
CIFAR-10	VGG-16	0.83	0.36	0.63	0.69
	VGG-19	0.90	0.31	0.53	0.73
	ResNet-50	0.80	0.22	0.27	0.63
SVHN	VGG-16	0.73	0.55	0.38	0.58
	ResNet-50	0.78	0.48	0.07	0.01

Table 5. MAE for predicting the robustness

Dataset	DNN Model	Robustness Measures				Mean of
		Robustness Predictor	Confidence Regression	LSA Regression	DSA Regression	Robustness
MNIST	LeNet-1	0.15	0.24	0.39	0.31	1.30
	LeNet-4	0.13	0.31	0.40	0.31	1.40
	LeNet-5	0.13	0.23	0.46	0.46	1.37
CIFAR-10	VGG-16	0.20	0.38	0.29	0.26	0.93
	VGG-19	0.07	0.77	0.13	0.10	0.32
	ResNet-50	0.05	140.52	0.09	0.07	0.18
SVHN	VGG-16	0.13	0.17	0.19	0.17	0.43
	ResNet-50	0.05	0.08	0.10	0.10	0.16

are generally linked to robustness. These measures include confidence and two variants of surprise adequacy, which are reasonable features for robustness prediction.

Confidence: A DNN-based classifier gives the probability of each class in the output layer. The class with the maximum probability is the prescribed classification result, which is also referred to as confidence, presenting how confident the DNN model is with regards to the classification result. In general, high confidence means that the DNN model would not change the classification result easily, so in some sense confidence reflects the robustness of the DNN model. It gives a convenient robustness score as it comes from the normal classification process without overhead.

Surprise Adequacy: Surprise Adequacy (SA) is proposed as a metric to measure the quality of the test set for the DNN model [12], which provides users the surprising degree of each test example.The surprising degree is measured by the behavior difference of the DNN model caused by the test example and the training data, which also indicates the robustness of the test example. The DNN model is programmed by the training data and the behavior space caused by the training data represents the learned logic inside the DNN model. If an input example is more surprising, it causes more behavior differences with the training data and is rarer for the DNN model, leading its classification result less confident. So an SA value also indicates the robustness value of a given example.

According to [12], the SA metric has two variants, i.e., the Likelihood-based Surprise Adequacy (LSA) and the Distance-based Surprise Adequacy (DSA). LSA estimates the probability density of a given input example with the training data using Kernel Density Estimation (KDE), which is defined as follow:

$$g(\boldsymbol{x}) = \frac{1}{|\alpha_l(T)|} \sum_{\boldsymbol{x}_i \in T} K(\alpha_l(\boldsymbol{x}) - \alpha_l(\boldsymbol{x}_i)) \tag{14}$$

where T is the training dataset, α_l is the vector of activation values for the selected layer l, $|\alpha_l(T)|$ is dimensionality of the neuron outputs in the training set, and K is the KDE function. LSA is defined as follow:

$$LSA(\boldsymbol{x}) = -\log(g(\boldsymbol{x})) \tag{15}$$

DSA measures the surprising degree through the Euclidean distance of the neuron output vectors generated by the a given input and examples in the training dataset, which provides the information about the closeness of the given input to the decision boundary. For a new test input $\boldsymbol{x}$ with the class C, let $\boldsymbol{x}_i$ be the nearest example of $\boldsymbol{x}$ with the same classification C, and $\boldsymbol{x}_j$ be the nearest example of $\boldsymbol{x}_i$ with the different classification other than C. $dist_i$ is the Euclidean distance of neuron outputs between $\boldsymbol{x}$ and $\boldsymbol{x}_i$, and $dist_j$ is the Euclidean distance of neuron outputs between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. Then DSA is defined as

$$DSA(\boldsymbol{x}) = \frac{dist_i}{dist_j} \tag{16}$$

We adopt the polynomial regression to predict the robustness with SA values as the input. Namely, we vary polynomials of degrees up to n and select the one with the best PCC value. (In the experiment, we set $n = 10$.) Formally, we have

$$r = w_1 c^n + w_2 c^{n-1} + ... + w_n c^1 + b \tag{17}$$

where r represents the robustness value, c is the confidence, DSA, or RSA, and $w_1, \cdots, w_n, b$ are the coefficients to be learned.

For LSA and DSA, as they compare the difference between the test input and the training data, we cannot use the examples in the original training dataset to compute the LSA and DSA. Hence, we use 5,000 test examples in the test set to compute LSA and DSA values respectively as the training set to train the polynomial regression model, and other 1,000 test examples as the test set to evaluate the polynomial regression model.

Table 4 and Table 5 show the PCC and the MAE results of the confidence regression model. The PCC values of the robustness predictor are higher than those of the confidence regression model, whereas The MAE values of the robustness predictor are lower than those of the confidence regression model. The results suggest that the robustness predictor is superior to the confidence regression model in predicting the robustness.

It is worth noticing that the MAE value of the ResNet-50 model for CIFAR-10 dataset is much larger the others, because most of the confidence values are 1.0. This shows that the confidence has a weaker relationship with the robustness. This is the inherent defect of DNN models as they could prescribe a fragile classification result very confidently.

For all the DNN models, PCC values of the robustness predictor are higher and MAE values are lower than both values of LSA and DSA, revealing that the robustness predictor outperforms the regression models based on LSA and DSA.

Table 6 presents the time cost of obtaining the robustness values of 5,000 examples through the C&W algorithm and the robustness predictor. The experiment for the C&W algorithm is conducted with the source code in [11]. We can see that the C&W algorithm needs several hours, but the robustness predictor only needs less than one second for the robustness calculation of 5000 examples. Most of the adversarial attack techniques aim to get the minimum perturbation by iterative algorithms, which means that the accurate robustness result needs a certain number of iterations, i.e., the high time cost. When a new input comes, the attack algorithm needs to execute again to calculate the robustness value for this input. In contrast, the proposed robustness predictor only needs a number of robustness values. The time cost of the training process is offline, which does not have any influence during the using process and can be accepted. When the new input comes, the regression model predicts the robustness based on the output from the penultimate layer of the new input. The time cost mainly comes from the DNN forward calculation, which is much less than the cost of optimization-based adversarial attack algorithms. Table 7 shows the memory and time cost of the robustness predictor for LeNet-5 on three types of android platforms. The robustness predictor can conveniently be applied on the mobile applications with the acceptable running cost. In conclusion, the robustness predictor is efficient for getting the robustness value.

For the ImageNet dataset, we trained a robustness predictor for a ResNet-50 model. Since the ImageNet is a large-scale image dataset with $224\times224\times3$ images and 1,000 classes, we increase the scale of the training data for the robustness predictor to 100,000 instances of neuron outputs from the penultimate layer. The architecture we design for the robustness predictor is $2048\times1000\times2048\times1024\times512\times256\times128\times10\times1$ fully connected neural network. Besides, we use the weights of 2048×1000 layer from the original ResNet-50 model to be the weights of the first layer (2048×1000) in the robustness predictor, and use the L_1 norm regularization in the 1000 layer. The results are shown in Table 8. The robustness predictor achieves the result of PCC 0.67 and MAE 39.46. The MAE shows the 34% additional errors with mean of robustness 116.64, which is the mean value of 1000 test examples' robustness computed by C&W. Compared to the result of confidence regression of PCC 0.61 and MAE 43.32, the robustness predictor works well on large-scale dataset. For the surprise-adequacy regression, it needs to compare the features of the test input and the training dataset, which is impossible for the large-scale ImageNet training dataset. As for the design of the robustness predictor, we suggest referring to the last few fully connected layers of the target DNN model. For the ResNet-50 model on ImageNet dataset, we use the last two layers in ResNet-50 as the first two layers to leverage the representations learnt by the target model. Since the dimension of representations of ResNet-50 on ImageNet are much higher than that on small datasets, we add more layers as dimension reduction and representation learning for the final regression task. During the practical usage of the robustness predictor for other datasets, users could first try the architecture of fully connected layers in the target DNN model, and then add layers for further dimension reduction and representation learning. According to our research, the predictor should not require a neural network with more than 10 layers.

Answer to RQ1: the robustness predictor can predict the robustness of individual input instances effectively and efficiently.

Table 6. Time cost (seconds) of the C&W adversarial attack measure and the robustness predictor

Dataset	DNN Model	C&W	Robustness Predictor
MNIST	LeNet-1	3,242.59	0.15
	LeNet-4	3,441.25	0.16
	LeNet-5	3,344.78	0.15
CIFAR-10	VGG-16	17,340.78	0.16
	VGG-19	13,116.62	0.16
	ResNet-50	44,129.12	0.16
SVHN	VGG-16	1995.18	0.07
	ResNet-50	4376.64	0.07

Table 7. Memory (MB) and time cost (ms) of the robustness predictor for LeNet-5 on android platforms

Type	Memory	Time Cost
Galaxy Nexus	31.38	21
Nexus 6	43.52	12
Pixel	44.48	10

Table 8. Results for ImageNet dataset

Robustness Measures	PCC	MAE	Mean of Robustness
Robustness Predictor	0.67	39.46	116.64
Confidence Regression	0.61	43.32	

4.3 (RQ2) Application range of the predictor

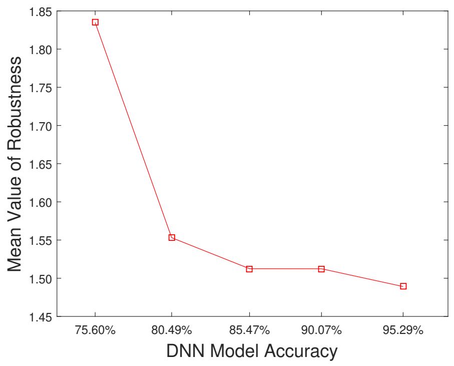


Fig. 4. DNN models' robustness with different accuracies

Since users may have different skills and experiences in training DNN models, not all the models can be trained well to reach high accuracy. We measure the mean robustness value of 1,000 examples in the MNIST test set for five LeNet-5 models with accuracies 75.60%, 80.49%, 85.47%, 90.07%, and 95.29%, respectively. The results are shown in Fig. 4.

In addition to the MNIST dataset, we also train three VGG-16 models for the CIFAR-10 dataset, with accuracies 71.71%, 82.16%, and 93.59% respectively. We observe that, for the same dataset and layer architecture, DNN models with different accuracies tend to have different performance on the robustness, with the same result for the CIFAR-10 dataset with the mean robustness values 0.96, 0.96 and 0.93 for the accuracies 71.71%, 82.16% and 93.59%.

We next examine whether the robustness predictor is effective for DNN models with different accuracies. For the evaluation, as before, we compare the result of the robustness predictor with those based confidence, LSA and DSA on DNN models. Fig. 5 and Fig. 6 show the evaluation results, where RP, CR, LSA and DSA standard for robustness predictor, confidence, LSA and DSA respectively.

For PCC, the robustness predictor obtains the values of no less than 0.9 for the MNIST dataset and no less than 0.7 for the CIFAR-10 dataset. For MAE, the robustness predictor achieves 0.1-0.17 and 0.20-0.22 for the MNIST and CIFAR-10 datasets. Both confirm the efficacy of our approach in handling models of different accuracies, and its superiority over other methods.

Answer to RQ2: the robustness predictor can be applied to DNN models with different accuracies.

4.4 (RQ3) Layer selection

Table 9. Result of layer selection on MNIST

Layer	PCC	MAE
Penultimate	0.95	0.13
Penultimate, dense_2	0.96	0.12
Penultimate, dense_2, pooling_2	0.96	0.13
Penultimate, dense_2, pooling_2, pooling_1	0.95	0.13

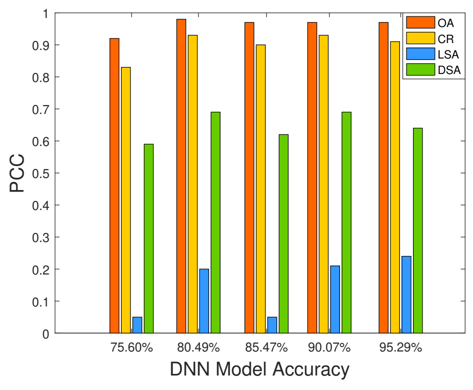
Table 10. Result of layer selection for VGG-16 on CIFAR-10

Layer	PCC	MAE
Penultimate	0.83	0.20
Penultimate, pooling_5	0.83	0.20
Penultimate, pooling_5, pooling_4	0.81	0.21
Penultimate, pooling_5, pooling_4, pooling_3	0.80	0.22

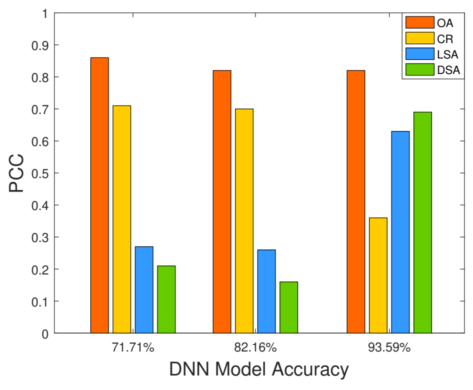
Table 11. Result of layer selection for ResNet-50 on CIFAR-10

Layer	PCC	MAE
Penultimate	0.80	0.05
Penultimate, pooling_5	0.77	0.06
Penultimate, add_16, add_15	0.79	0.06
Penultimate, add_16, add_15, add_14	0.79	0.06

Using the neuron output from the penultimate layer to train the robustness predictor is based on our intuition that the features in the deeper layers are more useful. However, it is important to study the impact of the selection of layers on the robustness predictor. To this end, we add neuron output from other layers to train the robustness predictor and compute the PCC and MAE results. In the experiment, we use three DNN

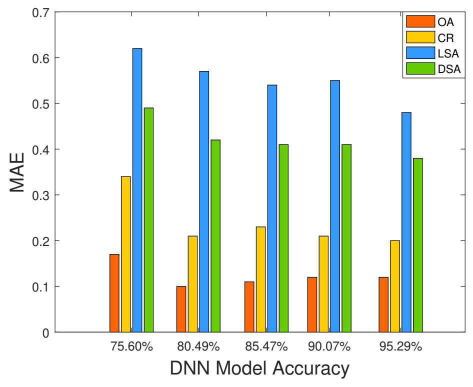


(a)

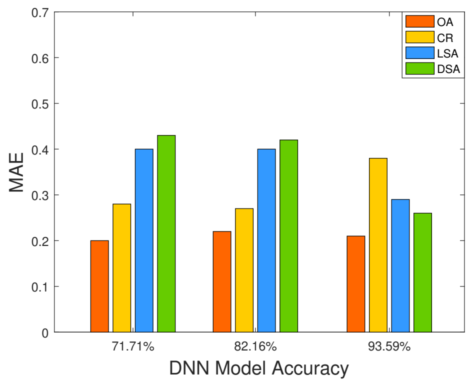


(b)

Fig.5. PCC results of DNN models with different accuracies



(a)



(b)

Fig.6. MAE results of DNN models with different accuracies

models, LeNet-5 for classifying the MNIST dataset, VGG-16 and ResNet-50 for classifying the CIFAR-10 dataset. As the number of neurons in some layers may be too large, we randomly select 100 neurons. (If there are no more than 100 neurons for a layer, all will be included.) As the input data becomes more complicated, we also add some layers the robustness predictor to improve its capacity. Specifically, we gradually add the neuron outputs from *dense*_2, *pooling*_2 and *pooling*_1 of LeNet-5 and the neuron outputs from *pooling*_5, *pooling*_4 and *pooling*_3 of VGG-16, *add*_16, *add*_15 and *add*_14 of ResNet-50 to the training data for the robustness predictor. For VGG-16, we choose the *pooling* layers, which are the last layers of the convolutional blocks. For ResNet-50, we choose the *add* layers, which are last layers of the residual blocks. As for the architecture, we use 120×100×100×84×10 fully connected layers for the robustness predictor of LeNet-5 and 256×256×256×256×10 fully connected layers for that of VGG-16 and ResNet-50. The results are shown in Table 8 and Table 9. No substantial difference is observed for both PCC and MAE. This suggests that adding extra neuron output from other layer may not further improve the performance of the robustness predictor. Namely, considering the neuron output from the penultimate layer gives sufficient features for the robustness prediction.

Answer to RQ3: there is no significant improvement *for the robustness predictor by adding neuron outputs from other layers.*

4.5 (RQ4) Effectiveness of the detector

To evaluate the effectiveness of the adversarial example detection, we compare the proposed detector with two state-of-the-art detection methods based on kernel density (KD) and local intrinsic dimensionality (LID), respectively.

We trained six DNN models (LeNet-4 and LeNet-5 for the MNIST dataset, VGG-16 and ResNet-50 for both SVHN and CIFAR-10 datasets) for evaluation. Following the experimental setting [42], the logistic regression detectors of KD and LID are trained by the scores of adversarial examples generated by FGSM. For the detector in our approach, we use the scores of adversarial examples generated by C&W. 1,000 adversarial examples generated by FGSM, C&W, BIM and JSMA respectively are used as the test set. We compare the AUROC results of the proposed detector with KD and LID based detectors as shown in Table 10.

We can see that the proposed detector obtains better performance than KD and LID detectors for C&W, BIM and JSMA, which are based on the optimization algorithms, on SVHN and CIFAR-10 datasets. For LeNet-5, the proposed detector outperforms the other methods for adversarial examples generated by the four adversarial attacks, the same for LeNet-4 except for the BIM. Based on the experimental results, we conclude that the proposed detector works effectively for detecting adversarial examples generated by those optimization algorithms.

Moreover, although the proposed detector has a weaker effect on the FGSM attack, it is still meaningful in detecting adversarial examples. Adversarial examples generated by FGSM are usually easy to detect, since the perturbation is relatively obvious, while adversarial examples generated by the optimization algorithms with imperceptible perturbation are difficult to detect, which is the strength of our approach.

Answer to RQ4: the detector can detect adversarial examples effectively.

4.6 Threats of Validity

In this section, we analyze threats to internal, external and construct validity for our work.

The selection of the datasets and the DNN models may externally threaten the validity of our evaluations. For this problem, we choose three popular datasets, including MNIST, CIFAR-10 and SVHN, and six state-of-the-art DNN models, including LeNet-1, LeNet-4, LeNet-5, VGG-16, VGG-19 and ResNet-50, as the experimental subjects. The datasets and the DNN models are diverse and representative for our experiments to draw the conclusions. The implementation in our experiments may also threaten the external validity of our conclusions. We depend on the widely used Python libraries, including Numpy, Sklearn, and Scipy to implement our code for experiments.

The internal threat of the validity might come from that we need a white-box DNN model for our approaches. Both the robustness predictor and the detector for detecting adversarial examples need the neuron outputs inside the target DNN model as the training data, which narrows the application range of our approaches. Usually, our approaches are used by users, who provide the DNN models. The predictor and the detector can be trained during the training process of the target DNN model and used as the auxiliary tools. Moreover, the robustness predictor is also a DNN-based model, so it would also suffer from the adversarial attack. Note that we provide the robustness predictor as the quality of service of the DNN model and strive to make the robustness prediction convenient to use and

Table 12. AUC (%) results for detecting adversarial examples

Dataset	DNN Model	Detector	Adversarial Attack			
			FGSM	C&W	BIM	JSMA
MNIST	LeNet-4	KD	81.97	97.52	90.91	98.52
		LID	89.12	99.69	98.36	99.06
		Ours	93.21	99.65	96.40	99.08
	LeNet-5	KD	72.70	89.86	75.65	94.61
		LID	78.68	97.78	92.01	93.26
		Ours	91.32	98.80	95.97	97.73
SVHN	VGG-16	KD	79.31	85.45	79.15	85.95
		LID	70.54	84.45	73.89	83.84
		Ours	78.20	97.63	82.76	97.43
	ResNet-50	KD	57.41	61.67	57.62	59.57
		LID	79.90	85.30	77.88	71.93
		Ours	78.18	92.85	82.39	92.77
CIFAR-10	VGG-16	KD	72.37	98.02	91.79	98.40
		LID	74.78	98.32	91.24	98.63
		Ours	72.82	98.44	95.70	98.78
	ResNet-50	KD	53.13	69.86	53.35	68.31
		LID	75.46	76.85	68.94	80.70
		Ours	66.13	97.86	71.90	94.30

less costly. In practice, the robustness predictor can be used in at least two ways. First, it predicts the robustness value as the quality measurement during the training process to, for instance, obtain a better DNN model. Second, it can be deployed as a part of the target DNN model to indicate the robustness of the prediction for a new input instance. Usually users are harder to acquire the internal neuron output of the target DNN model, so it would be much more difficult to synthesize the adversarial example for the robustness predictor.

As for the threat of construction validity, the metrics for evaluating the experimental results are important. We use PCC to measure the correlation between the results predicted by the regression model and the robustness values computed by the C&W attack, which is commonly applied to measure the linear correlation in, for instance, statistics and data mining. We use MAE to measure the absolute error between the prediction results and the robustness values, which is also a generic measurement for the numerical difference in machine learning research. We use AUC to measure the effectiveness of detecting adversarial examples, which is standard for comparison between classifiers in the machine learning.

5 Conclusion and Future Work

In this paper, we have proposed a fast robustness predictor, which is to predict a robustness measure for a new input example of DNN-based classifiers. The robustness predictor can be co-trained with the classifier. The lightweight feature of the approach makes it feasible to deploy on resource-constrained platforms such as mobile devices. Based on the framework of the robustness predictor, we also devised a detector for adversarial examples generated by optimization based attacks. The experimental results show the effectiveness and efficiency of the proposed robustness predictor and adversarial example detector.

In this paper, we only consider one type of perturbation, the L_2 norm distance metric and other robustness properties, such as light change, image rotation and so on, are also expected to be considered. Towards the different forms of the perturbation, neuron output solely may not be enough; Extra information, such as the neuron positions in the DNN model may be needed. These deserve further investigation. Furthermore, since we are able to use the neuron output and the DNN structure to predict the robustness of DNN models, it is conceivable one may apply to, e.g., DNN interpretation along this avenue. Since we only evaluate the proposed predictor on CNN models, another direction is to extend the scope of our approach to other types of DNN models, such as classic DNN and RNN models.

References

[1] Andor D, Alberti C, Weiss D, Severyn A, Presta A, Ganchev K, Petrov S, Collins M. Globally normalized transition-based neural networks. arXiv:1603.06042, 2016. https://arxiv.org/abs/1603.06042.

[2] Benesty J, Chen J, Huang Y, Cohen I. Pearson correlation coefficient. In *Noise reduction in speech processing*, Springer, pp.1-4.

[3] Bojarski M, Testa D D, Dworakowski D, Firner B, Flepp B, Goyal P, Jackel L D, Monfort M, Muller U, Zhang J. End to end learning for self-driving cars. arXiv:1604.07316, 2016. https://arxiv.org/abs/1604.07316.

[4] Carlini N, Wagner D. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2017, pp.39-57.

[5] Esteva A, Kuprel B, Novoa R A, Ko J, Swetter S M, Blau H M, Thrun S. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 2017, 542(7639): 115.

[6] Goodfellow I J, Shlens J, Szegedy C. Explaining and harnessing adversarial examples. arXiv:1412.6572, 2014, https://arxiv.org/abs/1412.6572.

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

[8] Hinton G, Deng L, Yu D, Dahl G, Mohamed A, Jaitly N, Senior A, Vanhoucke V, Nguyen P, Kingsbury B. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 2012, 29.

[9] Huang X, Kroening D, Kwiatkowska M, Ruan W, Sun Y, Thamo E, Wu M, Yi X. Safety and trustworthiness of deep neural networks: a survey. arXiv:1812.08342, 2018, https://arxiv.org/abs/1812.08342.

[10] Huang X, Kwiatkowska M, Wang S, Wu M. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification*, Springer, 2017, pp.3-29.

[11] Katz G, Barrett C, Dill D L, Julian K, Kochenderfer M J. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, Springer, 2017, pp.97-117.

[12] Kim J, Feldt R, Yoo S. Guiding deep learning system testing using surprise adequacy. In *Proceedings of the 41st International Conference on Software Engineering*, IEEE Press, 2019, pp.1039-1049.

[13] Krizhevsky A, Hinton G. Learning multiple layers of features from tiny images. Technical Report, Citeseer, 2009.

[14] Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 2012, pp.1097-1105.

[15] Kurakin A, Goodfellow I, Bengio S. Adversarial examples in the physical world. arXiv:1607.02533, 2016, https://arxiv.org/abs/1607.02533.

[16] LeCun L, Bengio Y, Hinton G. Deep learning. *Nature*, 2015, 521(7553): 436.

[17] LeCun L, Boser B, Denker J S, Henderson D, Howard R E, Hubbard W, Jackel L D. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1989, 1(4): pp.541-551.

[18] LeCun L, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998, 86(11): 2278-2324.

[19] Ma J, Sheridan R P, Liaw A, Dahl G E, Svetnik V. Deep neural nets as a method for quantitative structure–activity relationships. *Journal of chemical information and modeling*, 2015, 55(2): pp.263-274.

[20] Ma L, Juefei-Xu F, Zhang F, Sun J, Xue M, Li B, Chen C, Su T, Yang L, Liu Y. Deepgauge: Multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ACM, 2018, pp.120-131.

[21] Ma L, Zhang F, Xue M, Li B, Liu Y, Zhao J, Wang Y. Combinatorial testing for deep learning systems. arXiv:1806.07723, 2018, https://arxiv.org/abs/1806.07723.

[22] Moosavi-Dezfooli S, Fawzi A, Fawzi O, Frossard P. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp.1765-1773.

[23] Moosavi-Dezfooli S, Fawzi A, Frossard P. 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.

[24] Papernot N, McDaniel P, Goodfellow I, Jha S, Celik Z B, Swami A. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, 2017, pp.506-519.

[25] Papernot N, McDaniel P, Jha S, Fredrikson M, Celik Z B, Swami A. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, IEEE, 2016, pp.372-387.

[26] Pei K, Cao Y, Yang J, Jana S. Deepxplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp.1-18.

[27] Pulina L, Tacchella A. An abstraction-refinement approach to verification of artificial neural networks. In *International Conference on Computer Aided Verification*, Springer, 2010, pp.243-257.

[28] Pulina L, Tacchella A. Challenging SMT solvers to verify neural networks. *Ai Communications*, 2012, 25(2): 117-135.

[29] Ruan W, Huang X, Kwiatkowska M. Reachability analysis of deep neural networks with provable guarantees. arXiv:1805.02242, 2018, https://arxiv.org/abs/1805.02242.

[30] Rumelhart D E, Hinton G E, Williams R J. Learning representations by back-propagating errors. *Cognitive modeling*, 1988, 5(3): 1.

[31] Silver D, Huang A, Maddison C J, Guez A, Sifre L, Driessche G V D, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M. Mastering the game of Go with deep neural networks and tree search. Nature, 2016, 529(7587): pp.484.

[32] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556, 2014, https://arxiv.org/abs/1409.1556.

[33] Singh G, Gehr T, Püschel M, Vechev M. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 2019, 3(POPL): 41.

[34] Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp.2818-2826.

[35] Szegedy C, Zaremba W, Sutskever I, Bruna J, Erhan D, Goodfellow I, Fergus R. Intriguing properties of neural networks. arXiv:1312.6199, 2013, https://arxiv.org/abs/1312.6199.

[36] Weng T, Zhang H, Chen H, Song Z, Hsieh C, Boning D, Dhillon I S, Daniel L. Towards fast computation of certified robustness for relu networks. arXiv:1804.09699, 2018, https://arxiv.org/abs/1804.09699.

[37] Wu Y, Schuster M, Chen Z, Le Q V, Norouzi M, Macherey W, Krikun M, Cao Y, Gao Q, Macherey K. Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv:1609.08144, 2016, https://arxiv.org/abs/1609.08144.

[38] Xiang W, Tran H, Johnson T T. Output reachable set estimation and verification for multilayer neural networks. *IEEE transactions on neural networks and learning systems*, 2018, 29(11): 5777-5783.

[39] Xiong W, Droppo J, Huang X, Seide F, Seltzer M, Stolcke A, Yu D, Zweig G. Achieving human parity in conversational speech recognition. arXiv:1610.05256, 2016, https://arxiv.org/abs/1610.05256.

[40] Yuan Z, Lu Y, Wang Z, Xue Y. Droid-sec: deep learning in android malware detection. *ACM SIGCOMM Computer Communication Review*, 2014, 44(4): 371-372.

[41] Feinman R, Curtin R R, Shintre S, Gardner A B. Detecting adversarial samples from artifacts. 2017, arXiv:1703.00410, https://arxiv.org/abs/1703.00410.

[42] Ma X, Li B, Wang Y, Erfani S M, Wijewickrema S, Schoenebeck G, Song D, Houle M E, Bailey J. Characterizing adversarial subspaces using local intrinsic dimensionality. 2018, arXiv:1801.02613, https://arxiv.org/abs/1801.02613.

[43] Papernot N, McDaniel P, Jha S, Fredrikson M, Celik Z B, Swami A. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp.372-387.

[44] Netzer Y, Wang T, Coates A, Bissacco A, Wu B, Ng A Y. Reading digits in natural images with unsupervised feature learning. http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf, 2011.

[45] TensorFlow Lite. https://www.tensorflow.org/lite.

[46] PyTorch Mobile. https://pytorch.org/mobile/home/.

[47] Pang T, Du C, Dong Y, Zhu J. Towards robust detection of adversarial examples. In *Advances in Neural Information Processing Systems*, 2018, pp.4579-4589.

[48] Santhanam G K, Grnarova P. Defending against adversarial attacks by leveraging an entire gan. 2018, arXiv:1805.10652, https://arxiv.org/abs/1805.10652.

[49] Grosse K, Manoharan P, Papernot N, et al. On the (statistical) detection of adversarial examples. 2017, arXiv:1702.06280, https://arxiv.org/abs/1702.06280.

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

[51] Xu W, Evans D, Qi Y. Feature squeezing: Detecting adversarial examples in deep neural networks. 2017, arXiv:1704.01155, https://arxiv.org/abs/1704.01155.

[52] Wang X, Huang C, Yao L, Benatallah B, Dong M. A survey on expert recommendation in community question answering. *Journal of Computer Science and Technology*, 2018, 33(4), 625-653.

[53] Liu Q, Zhao H K, Wu L, Li Z, Chen E H. Illuminating Recommendation by Understanding the Explicit Item Relations. *Journal of Computer Science and Technology*, 2018, 33(4), 739-755.

[54] Ameur H, Jamoussi S, Hamadou A B. A new method for sentiment analysis using contextual auto-encoders. *Journal of Computer Science and Technology*, 2018 33(6), 1307-1319.

[55] https://github.com/geifmany/cifar-vgg

[56] Deng J, Dong W, Socher R, Li L J, Li K, Fei-Fei L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on Computer Vision and Pattern Recognition*, June 2009, pp.248-255.

[57] Li Z, Ma X, Xu C, Xu J, Cao C, Lü J. Operational calibration: debugging confidence errors for DNNs in the Field. 2019, arXiv:1910.02352, https://arxiv.org/abs/1910.02352.

[58] Li Z, Ma X, Xu C, Cao C, Xu J, Lü J. Boosting operational dnn 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*, 2019, pp. 499-509.

[59] LeCun Y, Bengio Y, Hinton G. Deep Learning. MIT Press, 2016.

[60] Bastani O, Ioannou Y, Lampropoulos L, Vytiniotis D, Nori A, Criminisi A. Measuring neural net robustness with constraints. In *Advances in neural information processing systems*, 2016, pp.2613-2621.

[61] Hendrycks D, Gimpel K. A baseline for detecting misclassified and out-of-distribution examples in neural networks. 2016, arXiv:1610.02136, https://arxiv.org/abs/1610.02136.

[62] Burrell J. How the machine 'thinks': Understanding opacity in machine learning algorithms. Big Data & Society, 2016, 3(1): 2053951715622512.

[63] Athalye A, Carlini N, Wagner D. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. 2018, arXiv:1802.00420, https://arxiv.org/abs/1802.00420.

[64] Wong E, Kolter J Z. Provable defenses against adversarial examples via the convex outer adversarial polytope. 2017, arXiv:1711.00851, https://arxiv.org/abs/1711.00851.

[65] Gopinath D, Pasareanu C S, Wang K, Zhang M, Khurshid S. Symbolic execution for attribution and attack synthesis in neural networks. In *IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion).*, 2019, pp.282-283.

[66] Wang Y, Li Z, Xu J, Yu P, Ma X. Fast Robustness Prediction for Deep Neural Network. In *Proceedings of the 11th Asia-Pacific Symposium on Internetware.*, 2019, pp.1-10.