Detecting Adversarial Samples for Deep Learning Models: A Comparative Study

Shigeng Zhang, *Member, IEEE*, Shuxin Chen, Xuan Liu, *Member, IEEE*, Chengyao Hua, Weiping Wang, Kai Chen, Jian Zhang, and Jianxin Wang, *Senior Member, IEEE*

***Abstract*—Deep learning techniques such as convolutional neural networks (CNNs) have been used in a wide range of fields due to their superior performance, e.g., image classification, autonomous driving and natural language processing. However, recent progress shows that deep learning models are vulnerable to adversarial samples, which are crafted by adding small perturbations on normal samples that are imperceptible to human beings but can mislead the deep learning models to output incorrect results. Many adversarial attack models are proposed and many adversarial detection methods are developed to detect adversarial samples generated by these attack models. However, the evaluations of these detection methods are fragmented and scatter in separate literature, and the community still lacks a comprehensive understanding of the ability and performance of existing adversarial detection methods when facing different attack models on different datasets. In this paper, by using image classification as the example application scenario, we conduct a comprehensive study on the performance of five mainstream adversarial detection methods against five major attack models on four widely used benchmark datasets. We find that the detection accuracy of different methods interleaves for different attack models and dataset. Moreover, besides detection accuracy, we also evaluate the time efficiency of different detection methods. The findings reported in this paper can provide useful insights when designing systems to detect adversarial samples and act as a guideline to design new methods to detect adversarial samples.**

***Index Terms*—Adversarial detection efficiency, adversarial samples detection, deep learning attacks, image classification.**

Manuscript received September 1, 2020; revised December 26, 2020; accepted January 24, 2021. Date of publication February 4, 2021; date of current version January 4, 2022. This work was supported in part by the National Key Research and Development Program (2020AAA0107800), the National Natural Science Foundation of China under Grants 61772559, 61602167, 61672543, U1734208, the Hunan Provincial Natural Science Foundation of China under Grant No. 2020JJ3016. Dr. Xuan Liu's work was partially supported by the National Defense Science and Technology Innovation Special Zone Project of China. Shunxin Chen's work was supported by the Fundamental Research Funds for the Central Universities of Central South University. Recommended for acceptance by B. Xiao. *(Corresponding authors: Xuan Liu and Jian Zhang.)*

Shigeng Zhang is with the School of Computer Science, and Engineering, Central South University, Hunan 410083, China, and also with the State Key Laboratory of Information Security (Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China (e-mail: sgzhang@csu.edu.cn).

Shuxin Chen, Chengyao Hua, Weiping Wang, Jian Zhang, and Jianxin Wang are with the School of Computer Science, and Engineering, Central South University, Hunan 410083, China (e-mail: 194712119@csu.edu.cn; chengyaohuawl@gmail.com; wpwang@csu.edu.cn; 308409399@qq.com; jxwang@csu.edu.cn).

Xuan Liu is with the College of Computer Science, and Electronic Engineering, Hunan University, Hunan 410082, China, also with the Science, and Technology on Parallel, and Distributed Processing Laboraty (PDL), 410073 (e-mail: xuan_liu@hnu.edu.cn).

Kai Chen is with the SKLOIS, the Institute of Information Engineering, Chinese Academy of Sciences, 100093, China, with the School of Cyber Security, University of Chinese Academy of Sciences, China, and also with the State Key Laboratory of Information Security (Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China (e-mail: chenkai@iie.ac.cn).

Digital Object Identifier 10.1109/TNSE.2021.3057071

I. Introduction

In recent years, deep learning models represented by convolutional neural networks (CNNs) have achieved excellent results in many fields, such as image classification [1], target detection [2], speech recognition [3] and natural language processing [4]. Although the performance of deep neural networks is better than human experts in many aspects, its security must be considered if we want to apply deep learning technology to safety critical scenarios in real life. Unfortunately, deep neural networks have been proven to be vulnerable to attacks from adversarial examples. As the application of neural networks becomes more and more widespread, it is particularly important to improve the robustness of deep neural networks against adversarial examples, especially in areas that require high safety such as medical diagnosis, criminal justice, identity recognition and autonomous driving. For example, as reported in [5], the attacker can modify the stop sign and the left turn sign so that the automatic driving system will mistakenly recognize them as the speed limit sign and the stop sign, respectively. Such adversarial attacks might cause serious traffic accidents. In [6], the authors show that an adversary can fool the person re-identification models to evade idenfication tracking or even impersonate a targeted different person in the physical world by wearing a carefully designed "invisible clock". In [7], the authors propose a generic and practical reconstruction attack against federated learning [8], which enables a malicious server to not only reconstruct the actual training samples, but also target a specific client and compromise the client-level privacy.

There have been many methods to generate adversarial examples against deep neural network models. From the perspective of the realization of the attacks, these methods can be divided into three categories: 1) gradient-based attacks, such as the fast gradient sign method (FGSM) [9], the basic iterative method (BIM) [10], the projected gradient descent (PGD) [11], the Jacobian-based saliency map attack (JSMA) [12], the Weighted JSMA (WJSMA) and the Taylor JSMA (TJSMA) [13];

2) optimization-based attacks such as Carlini & Wagner (C&W) [14]; and 3) decision-based attacks such as Deepfool [15]. In addition, from the perspective of whether the targeted label is specific or not, adversarial attacks can be classified into *targeted attacks* and *untargeted attacks*. In targeted attacks, the attacker generates adversarial examples that will be classified by the classifier as a specific category designated by the attacker. In untargeted attacks, adversarial examples might be misclassified as any categories rather than the correct cotegory.

In response to these adversarial attacks, researchers have proposed a series of adversarial attack defense methods. The proposed defense methods can also be divided into two categories. One type of method aims to generate robust models, and the other type of method aims to detect adversarial examples. The methods of generating robust models attempt to correctly classify adversarial examples. There are three ways to achieve this goal. The first direction is to modify model input data, including modifying training sample data and testing sample data, such as adversarial training [9] and data compression [16], [17]. The second direction is to modify the network structure and optimization techniques in order to enhance robustness to adversarial samples, including increasing the network depth and changing the loss function, such as defensive distillation [18], gradient obfuscation [19], and stability training [20]. The third direction is to add external modules to the original network model, such as feature squeezing [21]. However, these defenses techniques are not very effective against adversarial examples in classification tasks. For example, gradient obfuscation was once considered to be a very promising defense mechanism. However, in [22], an attack method that can effectively overcome the defense based on gradient ambiguity has been designed. In addition, these methods of generating robust models usually require high computational costs and lead to a decrease in the accuracy of the model on normal examples [23].

Considering the fact that it is difficult to generate a sufficiently robust model to correctly classify adversarial examples, detection-based methods have attracted great attention in recent years as another way to perform adversarial defense. Katzir *et al.* [24] pointed out that in different activation spaces, adversarial examples and normal samples have different spatial behaviors. Yang *et al.* [25] observed that the characteristic attribution graphs of adversarial examples are near the boundary and different from that of the corresponding original samples. Feinman *et al.* [26] used the kernel density estimation method to detect the adversarial examples far away from the data manifold in the hidden space of the final layer, and used Bayesian neural network uncertainty to detect adversarial examples that are close to the data manifold. Ma *et al.* [27] observed that the estimated local intrinsic dimensionality (LID) of adversarial examples is much higher than that of normal data samples, pointing out that LID can effectively capture the inherent dimensional properties of the adversarial area. Lee *et al.* [28] proposed a simple yet effective method for detecting any abnormal samples, which is claimed by the author to be applicable to any pre-trained softmax neural classifier. Through Gaussian discriminant analysis, the class conditional Gaussian distributions with respect to features of the deep models are obtained, and then a confidence score based on Mahalanobis distance is generated to distinguish adversarial samples from normal samples. These works detect adversarial samples generated by white-box attackes, i.e., assuming that the structure and parameters of the deep models are known to the attacker.

There are also some works on detecting adversarial samples generated by black-box attacks [29]. The success of black-box attacks stems from the transferability of adversarial samples, i.e., the effective adversarial samples against one model M_1 can also successfully attack another model M_2 trained on the same dataset. Pang *et al.* [30] proposed a new training objective function called reverse cross-entropy (RCE) to replace the commonly used ordinary cross-entropy (CE) loss. By minimizing RCE, the training procedure encourages the classifiers to return high confidence in the correct class while a uniform distribution on other classes for each sample. Compared to CE, the RCE training procedure can learn more distinguishable representations on detecting adversarial examples. Sperl *et al.* [31] supposed that adversarial examples provoke the dense layer neuron coverage of neural networks to behave in a distinctive pattern, which makes attacks detectable. They then proposed Dense-Layer-Analysis (DLA) to detect adversarial examples based on the behavior of neuron coverage when processing samples. These methods can effectively detect the adversarial samples generated under black-box attacks.

However, although many adversarial detection methods have been proposed, the community still lacks a comprehensive understanding on the ability and efficiency of these detection methods when facing different adversarial attacks on different datasets. The evaluations of adversarial detection methods scatter in separate literature and are fragmented. Actually, as to be shown in our experimental results, the performance comparison of different adversarial detection methods depends on the used dataset. It is highly probable that method A outperforms method B on dataset D1 but performs worse than B on another dataset D2. Moreover, most existing work only evaluates the detection accuracy of these methods but ignores the detection efficiency which is also very important in many real-time applications such as auto-driving. These facts motivate us to perform a comprehensively comparative study on the performance of mainstream adversarial detection methods. In this paper, we evaluate the ability and efficiency of five mainstream adversarial detection methods under five well-known attack models on four widely used benchmark datasets. The results provide a convincible evaluation to analyze the systematic robustness of different classification models.This makes the first step to build a holistic framework to evaluate and enhance the robustness of deep models in adversarial environments to ensure their safety when applied in real environments.

The findings reported in this paper can provide some insights when designing systems to detect adversarial samples and act as a guideline to design new methods to detect adversarial samples. We briefly describe our finds as follows. The detection ability of the ML-LOO detection algorithm [25] is

significantly stronger than other methods. It shows superior detection accuracy on a variety of datasets against a variety of attacks. However, because this method needs to process all the pixels of the image, it is very time-consuming. The KD +BU detection algorithm [26] requires least calculation, but it also has obvious defects. For example, for the C&W attack on CIFAR10, its detection accuracy is significantly worse than all other methods.

The rest of this paper is organized as follows. In Section II, we introduce some background knowledge of adversarial samples. Section III briefly introduces the five adversarial attack models considered in this paper. Section IV describes the detection methods to be evaluated in this paper and makes a comparison among them. Section V gives detailed experimental results of the five detection methods (described in Section IV) on the adversarial samples generated by the five attack models (described in Section III) on four mainstream benchmark datasets. Finally, Section VI closes the paper with some concluding remarks.

II. Background

A. Notations

Let $F(\cdot) : R^d \rightarrow [0,1]^C$ denotes a deep neural network classification model, which maps an image with dimension $d = h \times w \times c$ to the probability vector $F(x)$ with dimension C, where C is the number of categories (output labels). The input of $F(\cdot)$ is a vector representation of an image, and the output is the probability distribution of all possible classification labels, i.e., $F(x)_i$ represents the probability that the input vector x is classified as label i. Let $Z(\cdot)$ denote `Logits`, which is the output of the last layer of DNN (before the softmax layer). By using the softmax activation function on Logits to calculate the label probability, the model output can be written as

$$F(x) = softmax(Z(x)). \tag{1}$$

The predicted label $\hat{y}$ of x is the label corresponding to the maximum probability in $F(x)$, i.e.,

$$\hat{y} = \arg\max_i (F(x)_i). \tag{2}$$

Suppose that the number of layers of the deep neural network is $l + 1$, the input layer is the 0-th layer and the output layer is the l-th layer. We use $F^i(\cdot)$ to represent the output of the i-th neural network layer.

B. An Example of Adversarial Sample

The concept of adversarial sample was first introduced in [32]. Adversarial samples refer to samples formed by adding small perturbations to the original samples. These perturbations are imperceptible to human beings, but they can cause deep neural networks generating incorrect classification results with a high probability. Fig. 1 shows an example of adversarial sample. The adversarial attack only performs a small perturbation on the original image with the label of "orange," which can make the DNN to misclassify it as "cucumber" with a high degree of confidence.

Fig. 1. The original image (left) is classified as "orange," the small perturbations (middle) are added to the original image, and the adversarial image (right) is classified as "cucumber". The attack used in this example is FGSM. The shape of the original picture is $100 \times 100 \times 3$, and the l_2 distance between the original image and the confrontation image is 4.16.

C. Threat Models

The threat models in adversarial attacks can be divided into three categories [33]:

- The oblivious attacker does not know the existence of the detector *D* and generates adversarial examples based on the unsafe classification model *F*.
- The white-box attacker knows the existence of the detector *D*, and knows the scheme and parameters of the detector, and can design a special method to attack the model *F* and the detector *D* at the same time.
- The black-box attacker knows the existence of the detector *D* and how the detector is trained, but cannot know the parameters of the detector *D* and the model *F*.

III. Adversarial Attack Models

The algorithm or method used to generate adversarial samples is usually called an *adversarial attack model*. Given a model $F(\cdot)$ and a natural input image x, an adversarial sample $x' = x + \delta$ can be generated by adding a certain perturbation to x. Due to the effect of the perturbation δ, the predicted label of the adversarial sample x' by the model is not equal to the predicted label of its original sample x, i.e., $\hat{y}' \neq \hat{y}$, where

$$\hat{y}' = \arg\max_i (F(x')_i) \text{ and } \hat{y} = \arg\max_i (F(x)_i), \tag{3}$$

subject to that $||\delta|| < \epsilon$. Here $||\cdot||$ represents the distance metric and ϵ represents the maximum perturbation size, which restricts the perturbation amount so that the adversarial sample and the original sample are indistinguishable from the naked eyes. The distance metrics usually have three norms, namely l_0, l_2, and l_∞. For image data, the l_0 norm refers to the number of perturbed non-zero elements, which is the number of pixels modified by the adversarial attack in the original image. The l_2 norm is the standard Euclidean distance, which is the summation of the squares of the perturbed elements and then takes the square root. The l_∞ norm refers to the maximum value of each element of the perturbation. Recently, Wang *et al.* [34] used a new distance metric based on Just Noticeable Distortion (JND) to better measure the perceptual similarity and generate more perceptually realistic adversarial examples. In default, we use the l_2 norm to measure the size of the perturbation in this paper.

In recent years, many adversarial attack algorithms have been proposed. Here we introduce some well-known attack

TABLE I
SUMMARY OF ATTACK METHODS

Attack method	Concise Description	Learning Process	Strength
FGSM	Perturb each feature in the gradient direction of the model loss function with respect to the input vector.	One-shot	***
DeepFool	Iteratively perturb the image to generate adversarial samples. This method explores the nearest decision boundary.	Iterative	****
JSMA	Perturb a feature with a constant offset in each iteration step to maximize the saliency map.	Iterative	***
BIM	The iterative version of FGSM, multiple FGSM-like steps with smaller step sizes are adopted iteratively in the gradient direction.	Iterative	****
C&W	The attack is considered one of the most powerful attack algorithms to date. The basic idea is to introduce a cost function as a substitute instead of directly optimizing the loss function.	Iterative	*****

models that will be used to generate adversarial samples in our experimental evaluation. Table I gives a summary of these attack methods. Please note the strength of different attack models is evaluated according to the review given in [34] and the attack success rate we derived from our experiments. The attack models with more asterisks are considered to be more powerful.

A. Fast Gradient Sign Method (FGSM)

Goodfellow *et al.* [9] explained the basic principle of adversarial samples in 2014, pointed out that the intrinsic reason why adversarial samples exist is due to the high-dimensional linear property of the deep neural networks, and proposed a method for generating adversarial samples called the Fast Gradient Sign Method (FGSM). The idea of using FGSM to generate an adversarial sample from a normal sample can be represented as

$$x' = x + \varepsilon * sign(\nabla_x J(x, y)), \tag{4}$$

where x is the given original image, x' is the generated adversarial example image, J is the loss function, ε is used to limit the maximum size of the perturbation, and it is generally set to a value small enough such that it is difficult to distinguish with the naked eyes. Backpropagation is used to efficiently calculate the gradient. This method uses the gradient of the loss function to determine the direction of change of pixels, and eventually all the pixels will increase or decrease in equal proportions. For a variety of models, this method can effectively generate the required adversarial samples.

B. Basic Iterative Method (BIM)

Alexey *et al.* [10] proposed an iterative version of FGSM. The basic idea is to optimize the large-step operation to increase the loss function of the classifier by processing multiple small steps. It can increase the loss function and perform image perturbation to obtain the adversarial example with better adversarial resistance. The formula is

$$x'_i = clip_{x,\varepsilon}(x'_{i-1} + \varepsilon sign(\nabla_{x'_{i-1}} L(x'_{i-1}, y))), \tag{5}$$

where $x'_0 = x$, $clip_{x,\varepsilon}(\cdot)$ is the clipping function to keep x'_i within its range. In the iterative process, BIM adds many small perturbations along the maximum gradient, and after each perturbation, the gradient is recalculated. Compared with FGSM, BIM can attack more accurately because it omits some irrelevant perturbations, but the amount of calculation and time required also increase significantly.

C. Deepfool

DeepFool [15] is a non-target attack optimized for the l_2 norm distance metric, and it is also a white-box attack algorithm based on fast gradients. This method first assumes that the neural network is completely linear and the classification function of the classifier is $f(x) = w^T x + b$. According to the classification function, the hyperplane is $\Gamma = \{x : w^T + b = 0\}$. When a perturbation r is added at a certain point x_0 and it is perpendicular to the plane Γ, the added perturbation is the smallest and can meet the iteration requirements, as shown in the following formula

$$\begin{aligned} r^*(x_0) &= \arg\min_r ||r||_2 \\ s.t.\ sign(f(x_0 + r)) &\neq sign(f(x_0)), \end{aligned} \tag{6}$$

and there is a closed-form solution

$$r^*(x_0) = -\frac{f(x_0)}{||w||_2^2} w. \tag{7}$$

In the overall iterative process, the adversarial sample generation conforms to the following formula

$$\begin{aligned} &\arg\min_{r_i} ||r_i||_2 \\ s.t.\ &f(x_i) + \nabla f(x_i)^T r_i = 0. \end{aligned} \tag{8}$$

DeepFool uses iteration to generate the minimum norm to against the perturbation. At each step, it modifies the pixels located within the classification boundary to the outside boundary until a classification error eventually occurs. This method maintains almost the same A as FGSM, but produces less perturbation.

D. Jacobian-Based Saliency Map Attack

Nicolas *et al.* [12] proposed an attack method for non-cyclic feedforward neural networks, called *Jacobian-based Saliency Map Attack (JSMA)*, which generates perturbation by modifying a finite number of pixels in the original image. By using the forward derivative, the method uses the Jacobian matrix to calculate the characteristics from input to output

$$\nabla F(x) = \frac{\partial F(x)}{\partial x} = \left[\frac{\partial F_j(x)}{\partial x_i}\right]_{i \in 1...M, j \in 1..N}, \tag{9}$$

where F is the output of the last hidden layer of the classifier. The method of calculating the gradient of the preceding derivative is similar to backpropagation, but the preceding derivative directly uses the derivative of the network instead of the cost function. At the same time, the preceding derivative depends more on the characteristics of the input rather than the network parameters, so it can obtain more significant results in the output. The saliency mapping operation is performed on each forward derivative and output label, and the result is used as the basis for adversarial sample adjustment. The saliency mapping is as follows:

$$S(x,t)[i] = \begin{cases} 0, \text{ if } \frac{\partial F_t(x)}{\partial x_i} < 0 \text{ or } \sum_{j \neq t} \frac{\partial F_j(x)}{\partial x_i} > 0 \\ \left(\frac{\partial F_t(x)}{\partial x_i}\right) \left| \sum_{j \neq t} \frac{\partial F_j(x)}{\partial x_i} \right|, \text{otherwise} \end{cases}, \quad (10)$$

where i is the input feature. The $\frac{\partial F_t(x)}{\partial x_i}$ brought by the mapping requirement t is a positive number. When other points have a negative impact, the point that does not meet the relevant significance condition is set to 0. Finally, it takes the input feature that makes the largest of all the significant values to adjust the sample, and subtracts the original value to obtain the interference value. The purpose of constructing a saliency map is to study which pixel position changes have a greater impact on the target classification t (if the corresponding position derivative value is positive, increasing the pixel at the position can increase the score of the target t).

To perturb the input features through this method, it only needs to select pixels with large values for perturbation, which can quickly achieve the purpose of misclassification of the attacker. Compared with other methods, JSMA interferes with fewer pixels.

Recently, Loison *et al.* [13] proposed two probabilistic adversarial variants of JSMA, namely the *Weighted JSMA (WJSMA)* and the *Taylor JSMA (TJSMA)*. The main idea of WJSMA is to penalize gradients related to small probabilities to reduce their influence on saliency maps. The goal is to obtain a more balanced saliency map than that proposed by JSMA. The principle of TJSMA is to additionally penalize the choice of feature components that are close to the maximum feature value. These two methods can achieve significantly faster and more efficient attacks than JSMA.

E. Carlini & Wagner (C&W)

The C&W [14] attack is an optimization-based attack, and the adversarial samples produced by C&W are the ones with the smallest gap with the original samples among the five adversarial attacks models introduced in this section. Moreover, its attack power is considered the strongest. The essence is that in the iterative process, the discriminability of classification in the adversarial sample and the original sample are combined as a new optimization target to combine the iteration and optimization process. They define $x' = \frac{1}{2}(\tanh(w)+1)$ in terms of an auxiliary variable w, and solve

$$\min_w ||\frac{1}{2}(\tanh(w)+1) - x||_2^2 + c \cdot f\left(\frac{1}{2}(\tanh(w)+1\right), \quad (8)$$

where c is an introduced parameter. The C&W algorithm uses a dichotomy to find the smallest value of c. The objective function $f(\cdot)$ is defined as

$$f(x') = \max(\max\{Z(x')_i : i \neq t\} - Z(x')_t, -\mathrm{k}), \quad (12)$$

where k controls the confidence on adversarial samples. The value of k is the confidence values. When the value of k increases, the error probability of sample classification increases. Increasing the value of k makes the amount of calculation and the iteration time increase.

IV. Detection Methods

In this section, we summarize five mainstream adversarial detection methods to be evaluated in this paper, including the evaluation setup, results, pros and cons of each detection method. A summary of these detection methods is presented in Table II. The main results are the performance of detection methods against the oblivious attacker. Please not that the main results in the table are the performance of detection methods against the oblivious attacker.

A. Spatial Behavior in Activation Spaces

Katzir *et al.* [24] pointed out that normal samples and adversarial samples have different spatial behaviors in different activation spaces. By analyzing the activation spaces in the process of class separation, the spatial pattern that distinguishes adversarial samples and natural samples can be determined. The attacker designs the adversarial sample to be very similar to the corresponding normal sample but will cause the classifier to assign incorrect class labels to it. Therefore, in the activation space of the lower network layer, the author expects adversarial examples to appear in proximity to normal instances of their source class, while adjacent to instances of some other classes in the activation spaces of the last set of network layers.

The author first used the normal samples to construct the activation space, fed the normal samples into the network, and calculated the activation value of each layer. They then used principal component analysis (PCA) to reduce the dimensions of these activation values and construct a Euclidean activation space for each layer. After that, the author trained a dedicated k-NN classifier for each activation space and mapped the points in the space to a class label of the network. Due to the non-differentiability of the k-NN classification algorithm, it can provide gradient confusion, thereby preventing all simple gradient-based attacks on the detector. The author estimated the prior probability of class label conversion. The trained k-NN classifier is used to assign a class label sequence to each input sample, and each classifier assigns a label to the sample. The class label sequence of normal samples can be used to estimate the prior probability of class label changes between each pair of adjacent network layers. It is expected that this prior probability is relatively high in the first few activation spaces, and gradually decreases in the last few activation

TABLE II
SUMMARY OF DETECTION METHODS

Method	Summary	Datasets	Attacks	Main Results	Pros	Cons
SPBAS [24]	Use activation spaces to assign each input sample with a sequence of class labels and then computes the likelihood of the sequences	MNIST, CIFAR-10	C&W	MNIST AUC=0.91; CIFAR10 AUC=0.95	No need to use adversarial examples to train the detector	Only evaluating against the C&W attack; High memory requirements
ML-LOO [25]	Base on the significant difference in feature attributions between adversarial examples and original examples	MNIST, CIFAR-10, CIFAR-100	FGSM, PGD, C&W, BIM, JSMA	CIFAR-10 AUC: FGSM 0.997, PGD 0.999, C&W 0.995, JSMA 0.981, Deepfool 0.994	Evaluating on a diverse set of adversarial attacks; good performance against most attack	Query inefficiency; takes a long time to extract features
KD+BU [26]	Extract density estimated and Bayesian uncertainty estimates to construct two unsupervised detectors and also an combination of them	MNIST, CIFAR-10, SVHN	FGSM, C&W, BIM, JSMA	MNIST: Average AUC of 0.9259; CIFAR-10: Average AUC of 0.8554	Fast feature extraction; no need to use adversarial examples to train the detector	Low detector performance
LID [27]	characterizes the dimensional properties of adversarial regions via the use of Local Intrinsic Dimensionality	MNIST, CIFAR-10, SVHN	FGSM, C&W, BIM, JSMA	MNIST: Average AUC of 0.9756; CIFAR-10: Average AUC of 0.9189	Evaluating on a diverse set of adversarial attacks	The detection results will be affected by different minibatch sizes
MAHA [28]	Under Gaussian discriminant analysis, the confidence score of Mahalanobis distance is obtained based on the class conditional Gaussian distribution with upper and (low- and upper-level) features	SVHN, CIFAR-10, CIFAR-100	FGSM, BIM, DeepFool, C&W	CIFAR-100: FGSM 99.77%, BIM 96.90%, DeepFool 85.26%, C&W 91.77%	Suitable for any pre-trained softmax neural classifier, which can detect many attack methods	The number of samples available to train the logistic regression model is small

spaces, and the label conversion mode of adversarial samples will be very different from the label conversion mode of normal samples.

According to the prior probability of tag conversion, the likelihood of a label conversion sequence can be calculated. When calculating the likelihood, the authors follow the Naive Bayes principle and assume that the predictions of the k-NN classifier are independent of each other. When dealing with deep neural networks, because the number of layers of the network may be large, the label conversion sequence can become quite long. If the probabilities are multiplied, the precision of floating-point calculations will quickly be exhausted, resulting in large deviations in the results. To overcome this computational limitation, the author calculates the log likelihood (the sum of the logarithms of the probability) instead of the product. The author's process of processing adversarial samples is similar to the process of processing normal samples. The adversarial samples are input into the network, the corresponding activation values are calculated, and finally each adversarial sample is assigned a category label sequence. Finally, the authors calculate the likelihood of each normal sequence and adversarial sequence and choose a threshold to distinguish the adversarial samples from the normal ones.

B. ML-LOO: Detecting Adversarial Examples With Feature Attribution

Yang *et al.* [25] studied the application of feature attribution in detecting adversarial samples. The authors observed that the feature attribution map of the adversarial sample near the boundary is always different from the feature attribution map of the corresponding original sample. They also observed that this difference can be summarized by simple statistics that characterize differences in features, which can distinguish adversarial samples from natural samples. The authors speculated that this is due to the adversarial attack trying to perturb the sample to an unstable area on the decision plane.

The feature attribution method ϕ maps the input image $x \in R^d$ to the attribution vector $\phi(x) \in R^d$ that has the same shape as the image, so that the i-th dimension of $\phi(x)$ is the contribution of i-th feature to the model's prediction of a specific image. The authors focus on the leave-one-out method (LOO) [35], [36]. When the considered feature is covered by a certain reference value (such as 0), LOO assigns a reduced probability of the particular class to each feature. Denoting the example with the i-th feature masked by 0 as $x_{(i)}$, LOO defines ϕ as:

$$\phi(x)_i := F(x)_{\hat{y}} - F(x_{(i)})_{\hat{y}}. \tag{13}$$

The authors observe that there is a significant difference in the distribution of feature attribution values between normal samples and adversarial samples, so they propose to use interquartile range (IQR), which is the difference between the 75th percentile and the 25th percentile among all elements of the attribution vector $\phi(x)$, to detect adversarial samples. In order to detect adversarial samples with mixed confidence levels, the paper generalizes the proposed method to capture the dispersion of feature attribution outside the output layer of the model. For adversarial samples that are in a small neighborhood with the original sample in the pixel space but obtain a different category from the original sample with high

confidence in the output layer, the feature representation gradually deviates from the feature representation of the original sample along the layer. Therefore, the authors expect that the neurons in the middle layers contain uncertainties that can be captured by the feature attribution map. The paper denoted the map from input to an arbitrary neuron n of an intermediate layer of the model by $F_n : R^d \rightarrow R$. The feature attribution of neuron n is defined as $\phi_{F_n}(x) : R^d \rightarrow R^d$, such that the i-th entry quantifies the contribution of feature i to neuron n. For LOO, the authors indicate that

$$\phi_{F_n}(x)_i := F_n(x) - F_n(x_{(i)}). \tag{14}$$

C. Density Estimates and Bayesian Uncertainty Estimates

Using the intuition that adversarial samples are far away from the real data manifold, Feinman *et al.* [26] designed two indicators, *density estimation* and *Bayesian uncertainty estimation*, to detect adversarial samples. Density estimation is calculated using the training set in the feature space of the last hidden layer, and they are used to detect samples far away from the data manifold. Bayesian uncertainty estimation is used to detect when a sample is located in a low-confidence region of the input space, and it can detect adversarial samples that cannot be detection with the density estimation.

It is assumed that the technique of generating adversarial samples will usually generate an adversarial sample x' from the source sample x with prediction category $\hat{y}$. The adversarial samples are not on the manifold and are incorrectly classified as $\hat{y}'$. The paper models the submanifolds of each category by performing kernel density estimation in the feature space of the last hidden layer, and then detects adversarial samples. Let $h(x)$ be the activation vector of the last hidden layer of sample x, the density estimation of sample x with the prediction class $\hat{y}$ of is defined as

$$\hat{K}(x, X_{\hat{y}}) = \sum_{x_i \in X_{\hat{y}}} k_\sigma(h(x), h(x_i)), \tag{15}$$

where $X_{\hat{y}}$ is the set of training points of class $\hat{y}$, $k(\cdot, \cdot)$ is the kernel function defined in Eq. (16), and σ is the tuned bandwidth

$$k_\sigma(m, n) \sim \exp\Big(-\|m - n\|^2 \Big/ \sigma^2\Big). \tag{16}$$

Although the density estimation index can easily detect adversarial samples far away from the submanifold of $\hat{y}'$, when x' is very close to the submanifold, the effect of this index degrades. Therefore, the authors propose another indicator called Bayesian neural network uncertainty to overcome this limitation. For the adversarial sample and the prediction results of T times, the uncertainty estimate can be calculated as

$$U(x') = \frac{1}{T}\sum_{i=1}^{T} \hat{y}_i^{'T} \hat{y}_i^{'} - \left(\frac{1}{T}\sum_{i=1}^{T} \hat{y}_i^{'}\right)^T \left(\frac{1}{T}\sum_{i=1}^{T} \hat{y}_i^{'}\right). \tag{17}$$

The authors point out that when generating adversarial samples, the uncertainty is usually larger than the original sample and the density estimate is usually smaller. The reason is that the adversarial sample may be in an area with high uncertainty and/or low density estimation. Therefore, these two features are reliable indicators for detecting adversarial samples.

D. Local Intrinsic Dimensionality

Ma *et al.* [27] proposed to use *local intrinsic dimensionality* (LID) to detect adversarial samples. Given a reference sample $x \sim \mathrm{P}$, where P represents the data distribution, the estimator of LID at x is defined as

$$\widehat{LID}(x) = -\left(\frac{1}{k}\sum_{i=1}^{k} \log \frac{r_i(x)}{r_k(x)}\right)^{-1}, \tag{18}$$

where $r_i(x)$ represents the distance between x and its i-th nearest neighbor in the extracted point samples and $r_k(x)$ is the maximum value of the adjacent distance. For each sample and each conversion layer in the DNN, the LID estimate is calculated. The authors point out that the theoretical LID of adversarial samples is much larger than that of normal samples and thus can be used to detect adversarial samples.

E. Mahalanobis Distance

Lee *et al.* [28] proposed a simple and effective method that is suitable for any pre-trained softmax neural classifier. Under Gaussian discriminant analysis, for the upper and lower layer features of the deep neural network, the class conditional Gaussian distribution is obtained. The Mahalanobis distance (MAHA) is used to get the confidence score and build a logistic regression model based on which adversarial samples can be detected. First, on the pre-trained softmax neural classifier, a generative classifier that assumes the conditional-like Gaussian distribution obeys the multivariate Gaussian distribution is obtained, and the parameters of the generative classifier are estimated by calculating the empirical class mean and covariance of the training samples.

$$\begin{aligned} \hat{\mu}_{\mathrm{c}} &= \frac{1}{N_c}\sum_{i:y_i=c} f(x_i) \\ \hat{\Sigma} &= \frac{1}{N}\sum_{c}\sum_{i:y_i=c} (f(x_i) - \hat{\mu}_c)(f(x_i) - \hat{\mu}_c)^T, \end{aligned} \tag{19}$$

where N_c is the number of training samples with label c. This is equivalent to fitting the conditional-like Gaussian distribution with a tied covariance to training samples under the maximum likelihood estimator.

Second, using the above induced conditional-like Gaussian distribution, the Mahalanobis distance between the test sample x and the closest conditional-like Gaussian distribution is used to define the confidence score $M(x)$ as

$$M(x) = \max_c -(f(x) - \hat{\mu}_c)^T \hat{\Sigma}^{-1} (f(x) - \hat{\mu}_c). \tag{20}$$

In particular, the authors remark that abnormal samples can be characterized better in the representation space of DNNs.

Through the experiments of generating classifiers, they prove that the features of the trained DNNs support the hypothesis of GDA, which shows that the classifiers based on Mahalanobis distance are feasible to achieve softmax accuracy.

Then, in order to make in–distribution and out-of-distribution samples more separable, a small controlled noise is added to the test sample. The noise is generated to increase the proposed confidence score, as shown in Eq. (21). Moreover, by using the feature ensemble method to measure and combine the confidence scores of the final features in DNNs and other low-level features, the weight of each layer is obtained by training a logistic regression detector on the verification sample.

$$\begin{aligned}\hat{x} &= x + \varepsilon \cdot sign(\nabla_x M(x)) \\ &= x - \varepsilon \cdot sign(\nabla_x (f(x) - \hat{\mu}_{\hat{c}})^T \hat{\Sigma}^{-1} (f(x) - \hat{\mu}_{\hat{c}})). \end{aligned} \quad (21)$$

V. Detection Methods Evaluation and Comparison

We evaluated the five detection methods discussed in Section IV against the five attack models described in Section III on four benchmark datasets. In this section, we report the results and analyze them.

A. Experimental Setup and Datasets

Hardware Setup: All our experiments are conducted on a server equipped with an Xeon E5-2689 CPU and one GeForce RTX 2080Ti GPU. The memory capacity is 32 GB and the video memory size is 11 GB. The operating system is Linux Ubuntu 16.04.

Benchmark Dataset: We evaluate the considered detection methods on four widely used benchmark datasets: MNIST, SVHN, CIFAR10, and CIFAR100. The MNIST dataset consists of 70000 handwritten digits from 0 to 9, of which 60000 samples are used to construct the training set and 10000 samples are used to construct the testing set. Each image is represented by 28×28 gray pixels. The SVHN dataset consists of digital pictures in natural scenes. Its training set contains 73257 images, and its testing set contains 26032 images, each with a size of $32 \times 32 \times 3$. CIFAR10 is composed of 60000 color images, of which 10000 images are used to construct a testing set. Each image is stored by $32 \times 32 \times 3$ pixels. The CIFAR100 dataset has 20 categories, which are further divided into 100 sub-categories. Each sub-category contains 600 images (500 training images and 100 test images), and each image needs $32 \times 32 \times 3$ pixels to store.

B. Implementation of Attack/Detection Models

We use Keras with TensorFlow backend as our deep learning framework. For those attack/detection methods for which the code is available, we use the code provided by the authors. For those methods for which the code cannot be publicly obtained, we implement with Python and follow the settings given in corresponding papers. For MNIST and SVHN, we use a convolutional network composed of 32 convolutional layers. After each convolutional layer, there is a maximum pooling layer and a dropout layer with a loss rate of 0.25. The dropout layer used after each dense layer has a loss rate of 0.5. The classification accuracy of this classification network for MNIST and SVHN can reach 99.8% and 95.5%, respectively. For CIFAR-10 and CIFAR-100, we trained a 20-layer and 56-layer ResNet [37], and the recognition accuracy achieves 94.8% and 73.6%, respectively. The pixel values of the images in datasets are all scaled to the interval [0, 1]. The loss function used in each neural network training process is cross entropy. For the optimizer we choose the stochastic gradient descent (SGD) with a momentum of 0.9 and set the batch size at 128.

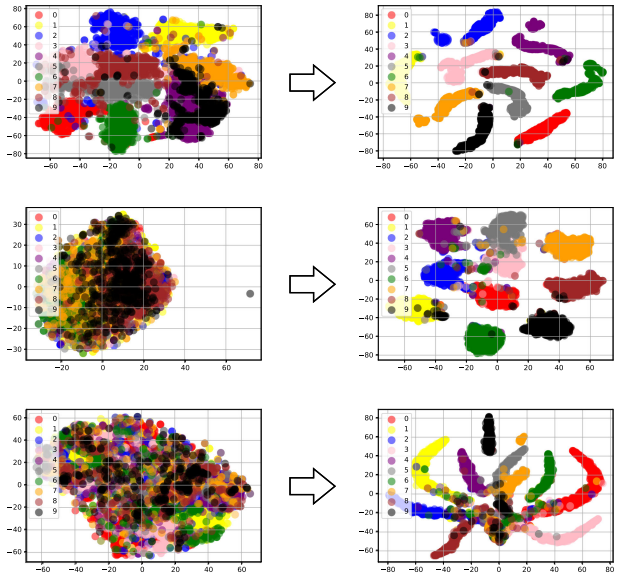


Fig. 2. The t-SNE visualization of the input space and output space of the classification network. The colors represent different class labels. The first, second, and third rows are visualizations of the MNIST, CIFAR10, and SVHN datasets, respectively. The first column represents the input space, and the second column represents the output space.

We use the t-distributed stochastic neighbor embedding (t-SNE) [38] to project the input space and the output space of the classification networks that we adopted for MNIST, CIFAR10, and SVHN, respectively, onto the 2D map. The results are shown in Fig. 2. We use different colors to represent different category labels. In the input space, the colors are interwoven and mixed, while in the output space, the colors are separated from each other. The obvious color separation shows the effectiveness of the classification model we use. The t-SNE maintains the relative distance of each data point to its neighbors in the projection visualization. It is an unsupervised dimensionality reduction method, especially suitable for the visualization of high-dimensional datasets.

C. Adversarial Examples and Noise Examples

We use cleverHans [39] and advertorch [40] to construct adversarial perturbations. According to the setting of the original paper for each adversarial detection method, we mainly test the performance of detection methods against the oblivious attacker, which is discussed in Section II-C. In addition,

TABLE III
THE MEAN L_2 PERTURBATION AND MODEL ACCURACY ON ADVERSARIAL EXAMPLES

	MNIST		CIFAR10		SVHN		CIFAR100	
	L_2	Acc.	L_2	Acc.	L_2	Acc.	L_2	Acc.
FGSM	8.40	0.13	12.63	0.06	12.63	0.08	16.63	0.01
C&W	1.84	0.00	0.13	0.00	3.39	0.00	0.12	0.00
BIM	5.28	0.02	8.05	0.00	9.83	0.02	8.99	0.00
JSMA	6.17	0.09	2.55	0.01	2.92	0.00	7.42	0.09
Deepfool	2.97	0.00	0.15	0.00	0.55	0.00	24.79	0.00

we focus on untargeted attacks, in which the attackers generate adversarial examples that might be misclassified by the classification network into any arbitrary incorrect label. For each dataset, we use each adversarial attack method to generate corresponding adversarial samples for 5000 test samples. We only attack the test samples correctly classified by the model because the attacker has no reason to attack the samples incorrectly classified by the classifier. Among them, 3000 adversarial images and 3000 corresponding normal images were used for the training of five adversarial sample detectors. We use the L_2 norm to represent the magnitude of the adversarial perturbations. The average perturbations size and the classification accuracy of the classifier on the adversarial samples are shown in Table III. The accuracy reflects the proportion of adversarial samples that were classified as the true label of their normal counterparts, and the smaller the value, the more effectiveness the attack is.

We report the detection results of these five detectors on the remaining 2000 adversarial images and corresponding normal images. For LID, KD+BU, and MAHA, we use both normal samples and noise samples when training the detector. Following the procedure outlined in [26], for JSMA attack, we first calculate the number of pixels that are disturbed when generating the adversarial samples and then generate the corresponding noise samples by changing the same number of pixels randomly selected in the original sample to the maximum value. For the other adversarial attacks, we generate noise samples by adding random noise to the normal samples. The magnitude of perturbation added to the noise samples is the same as the magnitude of perturbation added to the corresponding adversarial samples.

D. Feature Extraction

For SPBAS, we feed the normal samples and adversarial samples into the neural network and calculate the activation value, i.e., the output value of the neural network layer with the activation function. Due to the high dimensionality of these values, we use principal component analysis (PCA) to reduce the dimensionality of these activation values to construct a Euclidean activation space for the activation values. For neural network layers with more than 100 neurons, we reduce the dimensionality to 100 PCA components. For layers with fewer output neurons, we keep the original dimensionality. Then we train a dedicated k-NN classifier for each activation space and map the points in the space to a class label of the network. We set the value of k to 5. Then we use the trained k-NN classifier to assign a class label sequence to each input sample, where each classifier assigns a label to the sample.

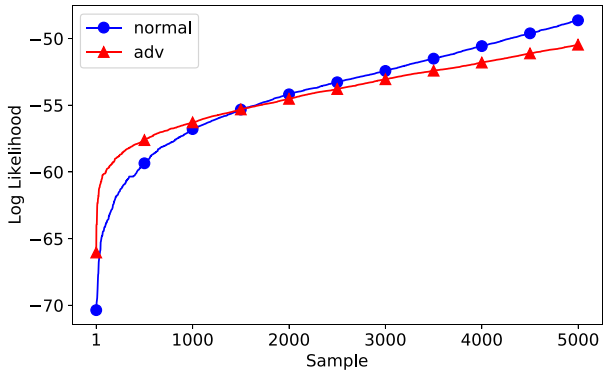


Fig. 3. Log likelihood of each normal sample and adversarial sample. To facilitate comparison, we sort the values in ascending order.

For ML-LOO, we select a set of layers in the middle of the neural network that use activation function as the layers we consider. In the output layer, we focus on the probability corresponding to the predicted label of the sample. We sequentially replace each pixel of the image with 0 and then record the IQR of the output change value of each neuron in the selected middle layer and the IQR of the probability change corresponding to the sample prediction label.

For KD+BU, we calculate the kernel density estimate in the feature space of the last hidden layer, as shown in Eq. (15). Furthermore, we set $T = 50$ to calculate the uncertainty estimate by 50 prediction results of sample x, as shown in Eq. (17).

For LID, we calculate the LID estimate for each network layer of each sample. The activation value of the neuron in a given layer is used as the input of the distance function required for this estimation. We use all network layers, including *conv2d*, *max-pooling*, *dropout*, *ReLU*, and *softmax*.

For MAHA, we consider measuring and combining the confidence scores of the final features in DNNs and the other low-priced features, calculating the empirical mean and tied covariance respectively by using the weighted average method to integrate them. We then use the validation samples to train a logistic regression detector to select the weight of each layer.

E. Detector Performance Comparison

We implement the detector in each detection method according to the detection framework described in corresponding original paper. For SPBAS, we use the k-NN classifier as the detector. For ML-LOO, KD+BU, LID, and MAHA, we use a logistic regression classifier as the detector. The adversarial samples are assigned as positive labels, and the normal samples and noise samples are assigned as negative labels. Since these classifiers are based on thresholds, we use the area under the ROC curve (AUC) as the major performance evaluation indicator. Because these classifiers are based on thresholds, we use the area under the ROC curve (AUC) as the major performance evaluation indicator, where the true positive rate (TPR) is the proportion of adversarial samples that

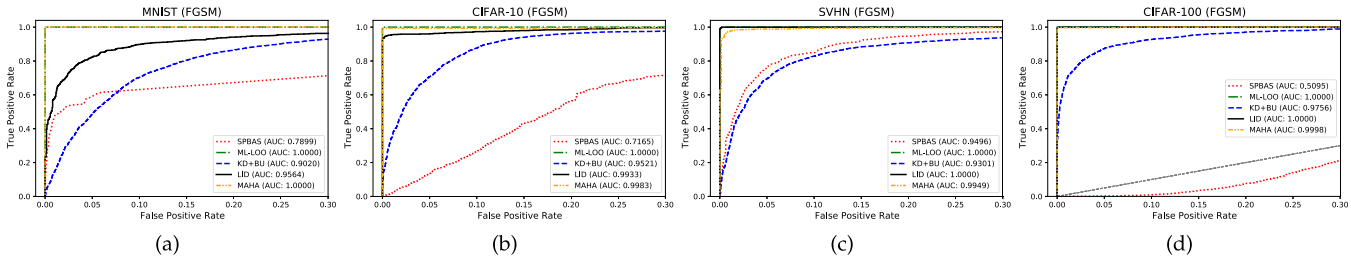


Fig. 4. ROC curves of five detection methods on four datasets against FGSM. We set the maximum distortion of adversarial example to 0.3.

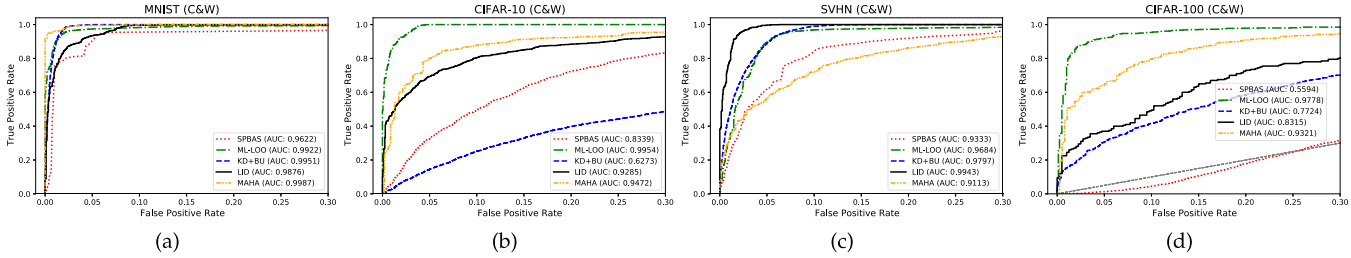


Fig. 5. ROC curves of five detection methods on four datasets against C&W. The confidence, iteration number, and binary search steps are set to 5, 1000, and 5, respectively.

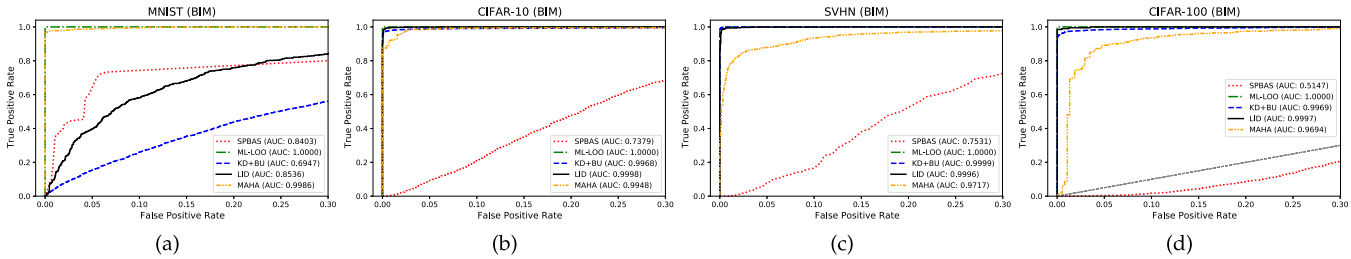


Fig. 6. ROC curves of five detection methods on four datasets against BIM. We set the maximum distortion of adversarial example and the step size for each attack iteration to 0.3 and 0.05, respectively.

are correctly classified as adversarial samples, and the false positive rate (FPR) is the proportion of normal samples or noisy samples that are misclassified as adversarial samples. We control the FPR between 0 and 0.3, because it is reasonable to control FPR in a relatively small range, and it can show the difference between ROC curves of various detection methods more clearly. We find that the detection accuracy of SPBAS on the CIFAR-100 dataset is low (approximately 0.5), which may be due to the fact that the likelihoods of the switching sequences of adversarial samples are almost the same as that of normal ones, as shown in Fig. 3. For this reason, we will not discuss the detection performance of SPBAS on CIFAR-100 in the following.

1) Detection Performance Against FGSM: The performance of the five detection methods under the FGSM attack model on the four datasets is given in Fig. 4. Generally, all the five detection methods perform fairly well in detecting adversarial samples generated by FGSM. ML-LOO performs the best on all the four datasets, while SPBAS and KD+BU perform worse than the other detection methods. Especially, the AUC value of SPBAS is only 0.7456 on the CIFAR-10 dataset, which means that it cannot effectively detect the FGSM attack on this dataset. Moreover, we can find that SPBAS performs better than KD+BU on MNIST and SVHN, but performs worse than KD+BU on CIFAR-10 and CIFAR-100. This means that the performance of different detection methods highly depends on the property of the datasets.

2) Detection Performance Against C&w: The performance of the five detection methods under the C&W attack model on the four datasets is given in Fig. 5. It can be observed that all the detection methods perform well on the MINST dataset, with all the AUC value higher than 0.97. However, the performance diverges on the other three datasets. For example, on the CIFAR-10 dataset, ML-LOO performs best with a high AUC value of 0.9954 while KD+BU performs worst with a low AUC value of only 0.6273. The performance flips among different detection methods are also observed, similar as for the FGSM attack. Generally, C&W seems to be slightly more complicated than FGSM because the average AUC is slightly lower.

3) Detection Performance Against BIM: The performance of the five detection methods under the BIM attack model on the four datasets is given in Fig. 6. It can be observed that SPBAS performs poorly in the detection of BIM on CIFAR-10 and CIFAR-100, with an AUC value of 0.7076 and 0.8200, respectively. In contrast, ML-LOO always performs the best

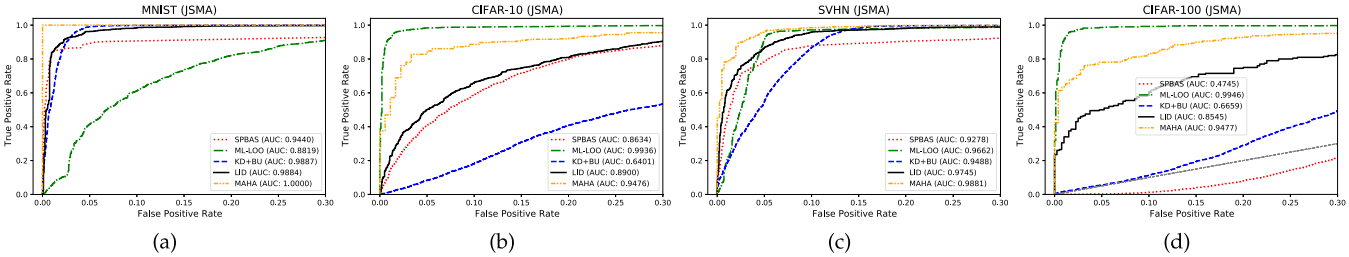


Fig. 7. ROC curves of five detection methods on four datasets against JSMA. The maximum percentage of perturbed features is set to 0.1.

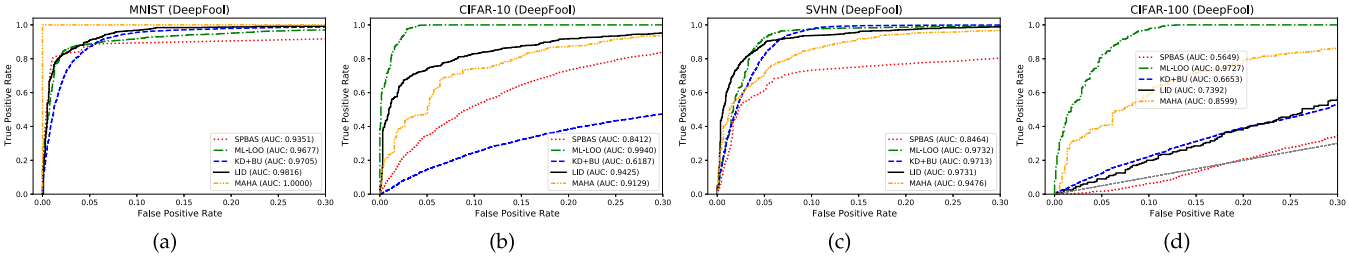


Fig. 8. ROC curves of five detection methods on four datasets against Deepfool. We set the maximum number of iteration to 50.

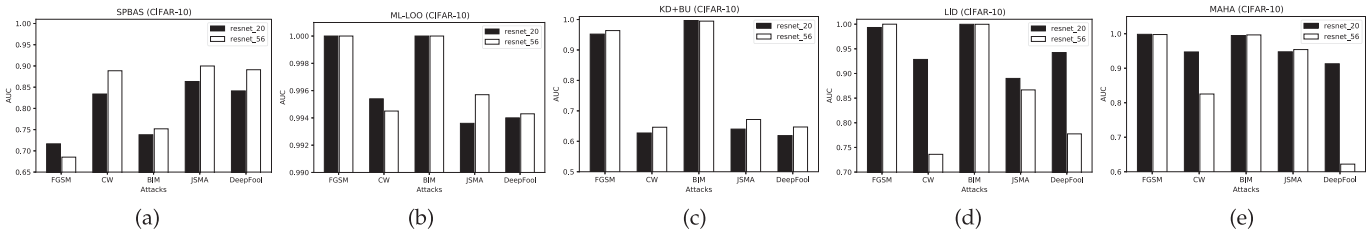


Fig. 9. Performance comparison of each detection method on resnet-20 and resnet-56 models against five attacks on CIFAR-10.

among the five detection methods with an AUC value of 1.00 on all the four datasets. KD+BU also performs poorly on the MNIST dataset, with the AUC value of only 0.6947.

4) Detection Performance Against JSMA: The performance of the five detection methods under the JSAM attack model on the four datasets is given in Fig. 7. Compared with FGSM and BIM, the adversarial samples generated by using JSAM are more difficult to detect. For example, the AUC value of KD+BU on the CIFAR-10 and CIFAR-100 dataset are only 0.6401 and 0.6659, respectively. It is worth pointing out that ML-LOO performs fairly poorly on the MNIST dataset, with the AUC value of only 0.8819. As a comparison, the AUC value of ML-LOO is higher than 0.96 in all other cases.

5) Detection Performance Against Deepfool: The performance of the five detection methods under the Deepfool attack model on the four datasets is given in Fig. 8. Similarly as in previous scenarios, ML-LOO performs the best and achieves the highest AUC value in three datasets (CIFAR-10, SVHN, CIFAR-100). On the MNIST dataset, the SPBAS performs the best with the AUC value of 1.00. KD+BU performs the worst with achieving the lowest AUC value in CIFAR-10 and CIFAR-100, both smaller than 0.67.

6) Detection Performance on Different Network Models: We also explore the detection performance of different detection methods for network modelds with different depth against the same dataset and attack to investigate the complexity of network model on the detection accuracy. We trained two models, namely ResNet-20 and ResNet-56, having 20 layers and 56 layers respectively, on the same dataset CIFAR-10. The results are shown in Fig. 9. It can be observed that the network depth does not have a consistent impact on detection accuracy. In some cases the adversarial samples generated for deeper models (ResNet-56) are easier to detect, but there are also some cases in which the adversarial samples generated for shallower models (ResNet-20) are easier to detect. For example, ML-LOO, LID, and MAHA are easier to detect C&W attacks on shallow networks, while the cases of SPBAS and KD+BU are opposite.

7) Summarization on Detection Accuracy: To summarize, we find that no detection method can consistently outperform other methods in detecting all the five attacks considered in this paper on the four datasets. The detection performance of ML-LOO is generally good, but as to be discussed in Section V-F, this detection method takes the most time, and thus its detection time efficiency is the lowest. Generally, how to design time-efficient methods that can be used to detect adversarial samples in generic cases is still an important issue to be solved in the future.

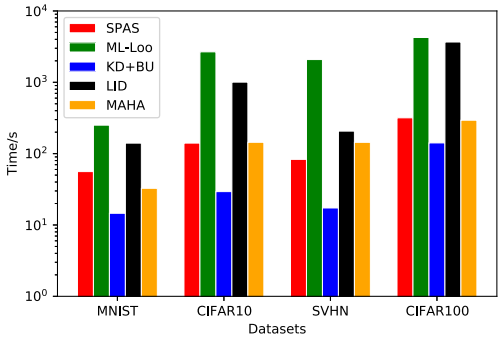


Fig. 10. The detection time statistics of five detection methods against five attacks on four datasets. The y-axis uses a logarithmic scale.

F. Detection Time Comparison

In order to detect adversarial samples, we need to first extract features, which varies with different detection methods, and then use these features to train a classifier to determine whether a sample is adversarial or not. The detection time thus consists of two parts: the time spent in feature extracting $(T_{extract})$ and the time spent in adversarial judging (T_{judge}). Because the capabilities of different attack models vary from one to another, the number of adversarial samples that can successfully achieve the attack is different. This fact leads to a varied number of adversarial samples we used to evaluate each attack. For a fair comparison, we calculated the average detection time of each method over 5000 samples on each dataset against each attack as

$$T = \frac{T_{extract} + T_{judge}}{N} \times M \tag{22}$$

where N represents the total number of samples used to evaluate an adversarial attack against a dataset, including normal samples, adversarial samples and noise samples (if necessary). Because the number of effective adversarial samples is close to 5000, we set $M = 5000$ in our evaluation. The detection time of the same detection method for five different attacks on the same dataset is relatively close, so we take the average over different datasets and report the average value. Because the magnitude of each detection method is significantly different, we use the logarithmic scale. The results are plotted in Fig. 10. We found that the ML-LOO detection method takes the most time. This might be because it needs to process every pixel of an image and to calculate the attribute change of each pixel after being covered by the reference value. It also needs the relevant information of the middle layer of the model. In contrast, KD + BU only computes the required information in the feature space of the last hidden layer. We also found that the more complex data sets require longer detection time, due to the increase in the number of pixels to be processed and the increase in the depth of the classification model used. In addition, in all the four datasets, the ranking of the five detection methods in detection time remains unchanged, which means that among the five detection methods, no detection method is particularly good at detecting adversarial samples generated on a specific data set.

VI. Conclusion

In this paper, we use image classification as an application scenario to study the detection performance of different adversarial sample detection methods in different data sets against different adversarial attacks. We conducted a large number of experiments on the four famous image classification data sets of MNIST, SVHN, CIFAR-10, and CIFAR-100, and compared the detection performance and time performance of SPBAS, ML-LOO, KD+BU, LID, and MAHA against common adversarial attacks. Through the analysis of the comparison results, we found that how to detect the existence of adversarial samples completely is still an open question.

In the future, we will evaluate the performance of various detection methods in more adversarial environments, for example, evaluating the detection performance of various detection methods against adversarial attacks with different confidence levels. We will gradually implement a framework that integrates various attack methods to evaluate the performance of different adversarial detection methods.

References

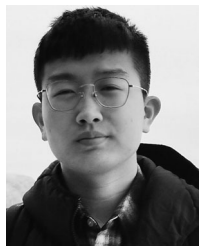
[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 3rd Int. Conf. Learn. Representation*, 2015, *arXiv:1409.1556*.
[2] A. Krizhevsky, I. Sutskever, and Geoffrey E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 9, no. 6, pp. 84–90, 2017.
[3] G. Hinton *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.
[4] M Bates, "Models of natural language understanding," in *Proc. Nat. Acad. Sci. USA*, vol. 92, no. 22, pp. 9977–9982, Oct. 1995.
[5] K. Eykholt *et al.*, "Robust physical-world attacks on deep learning visual classification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 1625–1634.
[6] Z. Wang, S. Zheng, M. Song, Q. Wang, A. Rahimpour, and H. Qi, "advPattern: Physical-world attacks on deep person re-identification via adversarially transformable patterns," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 8340–8349.
[7] M. Song *et al.*, "Analyzing user-level privacy attack against federated learning," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 10, pp. 2430–2444, Oct. 2020.
[8] B. McMahan, E. Moore, D. Ramage, S. Hampson, and Blaise Agüera y Arcas, "Communication-Efficient learning of deep networks from decentralized data," in *Proc. 20th Int. Conf. Artif. Intell. Statist.*, 2017, pp. 1273–1282.
[9] Ian J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proc. 3rd Int. Conf. Learn. Representation*, 2015, pp. 7–9.
[10] A. Kurakin, Ian J. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *Proc. 5th Int. Conf. Learn. Representation*, 2017, *arXiv:1607.02533*.
[11] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *Proc. 6th Int. Conf. Learn. Representat.*, 2018, *arXiv:1706.06083*.
[12] N. Papernot, Patrick D. McDaniel, S. Jha, M. Fredrikson, Z. Berkay Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Proc. IEEE Euro. Symp. Secur. Privacy*, 2016, pp. 372–387.

[13] A. Loison, T. Combey, M. Faucher, and H. Hajri. "Probabilistic jacobian-based saliency maps attacks," *Mach. Learn. Knowl. Extr.*, vol. 2, no. 4, pp. 558–578, 2020.
[14] N. Carlini and D. A. Wagner, "Towards evaluating the robustness of neural networks," in *Proc. IEEE Symp. Secur. Privacy*, 2017, pp. 39–57.
[15] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: A simple and accurate method to fool deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.,* 2016, pp. 2574–2582.
[16] G. K. Dziugaite, Z. Ghahramani, and D. M. Roy, "A study of the effect of JPG compression on adversarial images," *CoRR*, 2016, *arXiv:1608.00853*.
[17] N. Das *et al.*, "Keeping the bad guys out: Protecting and vaccinating deep learning with jpeg compression." *CoRR*, 2017, *arXiv:1705.02900*.
[18] N. Papernot, P. D. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *Proc. IEEE Symp. Secur. Privacy*, 2016, pp. 582–597.
[19] N. Papernot, P. D. McDaniel, Ian J. Goodfellow, S. Jha, Z. Berkay Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2017, pp. 506–519.
[20] S. Zheng, Y. Song, T. Leung, and I. J. Goodfellow, "Improving the robustness of deep neural networks via stability training," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 4480–4488.
[21] W. Xu, D. Evans, and Y. Qi. "Feature Squeezing: Detecting adversarial examples in deep neural networks," in *Proc. Annu. Netw. Distrib. Syst. Security Symp.*, 2018.
[22] A. Athalye, N. Carlini, and David A. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 274–283.
[23] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry, "There is no free lunch in adversarial robustness (but there are unexpected benefits)," *CoRR*, 2018, *arXiv: 1805.12152*.
[24] Z. Katzir and Y. Elovici, "Detecting adversarial perturbations through spatial behavior in activation spaces," in *Proc. Int. Joint Conf. Neural Netw.,* 2019, pp. 1–9.
[25] P. Yang, J. Chen, C.-J. Hsieh, J.-L. Wang, and M. I. Jordan, "ML-LOO: Detecting adversarial examples with feature attribution," in *Proc. 34th AAAI Conf. Artif. Intell., 32nd Innov. Appl. Artif. Intell. Conf., and 10th AAAI Symp. Edu. Adv. Artif. Intell.*, 2020, pp. 6639–6647.
[26] R. Feinman, R. R. Curtin, S. Shintre, and A. B. Gardner, "Detecting adversarial samples from artifacts," *CoRR*, 2017, *arXiv: 1703.00410*.
[27] X. Ma *et al.*, "Characterizing adversarial subspaces using local intrinsic dimensionality," in *Proc. Int. Conf. Learn. Representation*, 2018.
[28] K. Lee *et al.*, "A simple unified framework for detecting out-of-distribution samples and adversarial attacks," in *Adv. Neural Infor. Process. Syst. 31st: Ann. Conf. Neural Infor. Process. Syst.*, 2018, pp. 7167–7177.
[29] N. Papernot, Patrick D. McDaniel, Ian J. Goodfellow, S. Jha, Z. Berkay Celik, and A. Swami, "Practical black-Box attacks against deep learning systems using adversarial examples," *CoRR*, 2016, *arXiv: 1602.02697*.
[30] T. Pang *et al.*, "Towards robust detection of adversarial examples," *Adv. Neural Infor. Process. Syst. 31st: Annu. Conf. Neural Infor. Process. Syst.*, 2018, pp. 4584–4594.
[31] P. Sperl, C.-Y. Kao, P. Chen, X. Lei, and K. Böttinger, "DLA: Dense-Layer-Analysis for adversarial example detection," in *Proc. IEEE Eur. Symp. Secur. Privacy*, 2020, pp. 198–215.
[32] C. Szegedy *et al.* "Intriguing properties of neural networks," in *Proc. 2nd Int. Conf. Learn. Representation*, 2014, pp. 14–16.
[33] N. Carlini and D. A. Wagner, "Adversarial examples are not easily detected: Bypassing ten detection methods," in *Proc. 10th ACM Workshop Artif. Intell. Secur.*, 2017, pp. 3–14.
[34] Z. Wang, M. Song, S. Zheng, Z. Zhang, Y. Song, and Q. Wang, "Invisible adversarial attack against deep neural networks: An adaptive penalization approach," *IEEE Trans. Dependable Secure Comput.*, to be published, doi: 10.1109/TDSC.2019.2929047.
[35] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Proc. Comput. Vis. 13th Eur. Conf. Lecture Notes Comput. Sci.*, 2014, pp. 818–833.
[36] J. Li, W. Monroe, and D. Jurafsky. "Understanding neural networks through representation erasure," *CoRR*, 2016, *arXiv:1612.08220*.
[37] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Proc. Comput. Vis. Eur. Conf.*, 2016, pp. 630–645.
[38] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, no. 2605, pp. 2579–2605, 2008.
[39] I. J. Goodfellow, N. Papernot, and P. D. McDaniel. "Cleverhans v0.1: An adversarial machine learning library," *CoRR*, 2016, *arXiv:1610.00768*.
[40] G. W. Ding, L. Wang, and X. Jin. "Advertorch v0.1: An adversarial robustness toolbox based on Pytorch," *CoRR*, 2019, *arXiv: 1902.07623*.

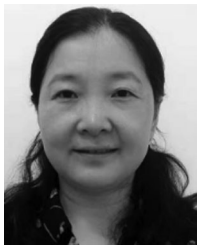
Shigeng Zhang (Member, IEEE) received the B.Sc., M.Sc., and D.Eng. degrees in computer science from Nanjing University, Nanjing, China, in 2004, 2007, and 2010, respectively. He is currently an Associate Professor with the School of Computer Science and Engineering, Central South University, Changsha, China. He has authored or coauthored more than 70 technique papers in top international journals and conferences including Ubicomp, Infocom, Mobihoc, ICNP, TMC, TC, TPDS, TOSN, and JSAC. His research interests include Internet of Things, mobile computing, RFID systems, and IoT security. He is on the editorial board of *International Journal of Distributed Sensor Networks*, and was a Program Committee Member of many international conferences including ICC, ICPADS, MASS, UIC, and ISPA. He is a Member ACM.

Shuxin Chen received the B.E. degree in computer science and technology from Xiangtan University, Xiangtan, China, in 2019. He is currently working toward the master's degree with the School of Computer Science and Engineering, Central South University, Changsha, China. His research interests include AI security and adversarial machine learning.

Xuan Liu (Member, IEEE) received the B.Sc. degree in information and computing mathematics from XiangTan University, Xiangtan, China, the M.Sc. degree in computer science from the National University of Defense Technology, Changsha, China, and the Ph.D. degree from The Hong Kong Polytechnic University, in 2005, 2008 and 2015, respectively. He is currently an Associate Professor with the College of Computer Science and Electronic Engineering, Hunan University, Changsha, China. She has authored or coauthored more than 30 technique papers in top international journals including JSAC, ToN, TMC, TC, and TPDS and top conferences including Infocom, ICNP, Mobihoc, and Ubicomp. Her research interests include Multiagent reinforcement learning, RFID systems, and Internet of Things.



Chengyao Hua received the B.E. degree in computer science and technology from Guangxi University, Nanning, China, in 2020. He is currently working toward the master's degree with the School of Computer Science and Engineering, Central South University, Changsha, China. His research interests include AI security and adversarial machine learning.



Weiping Wang received the Ph.D. degree in computer science from Central South University, Changsha, China, in 2004. She is currently a Professor with the School of Information Science and Engineering, Central South University. She has authored or coauthored more than 60 papers in refereed journals and conference proceedings. Her current research interests include network coding and network security.

Kai Chen received the Ph.D. degree from the University of Chinese Academy of Science, Beijing, China, in 2010. He is currently a Professor with the Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China, and a Professor with the University of Chinese Academy of Sciences, Beijing, China. His research interests include software analysis and testing, smartphones, and privacy.

Jian Zhang received the B.Eng. degree in computer science from the National University of Defense Technology, Changsha, China, the M.Eng. and Ph.D. degrees in computer science from Central South University, Changsha, China, in 1998, 2002, and 2007, respectively. He is currently an Associate Professor with the School of Computer Science and Engineering, Central South University. His research interests include optimization theory, cyberspace security, cloud computing, and cognitive radio technology.

Jianxin Wang (Senior Member, IEEE) received the B.Eng., M.Eng. degrees in computer engineering, and the Ph.D. degree in computer science from Central South University, Changsha, China, in 1992 and 1996, 2001, respectively. He is the Dean and a Professor with School of Computer Science and Engineering, Central South University. His current research interests include algorithm analysis and optimization, parameterized algorithm, bioinformatics, and computer network.