Adversarial Sample Detection for Deep Neural Network through Model Mutation Testing

Jingyi Wang*, Guoliang Dong†, Jun Sun*, Xinyu Wang†, Peixin Zhang†
*Singapore University of Technology and Design
†Zhejiang University

***Abstract*—Deep neural networks (DNN) have been shown to be useful in a wide range of applications. However, they are also known to be vulnerable to adversarial samples. By transforming a normal sample with some carefully crafted human non-perceptible perturbations, even highly accurate DNN makes wrong decisions. Multiple defense mechanisms have been proposed which aim to hinder the generation of such adversarial samples. However, a recent work show that most of them are ineffective. In this work, we propose an alternative approach to detect adversarial samples at runtime. Our main observation is that adversarial samples are much more sensitive than normal samples if we impose random mutations on the DNN. We thus first propose a measure of 'sensitivity' and show empirically that normal samples and adversarial samples have distinguishable sensitivity. We then integrate statistical model checking and mutation testing to check whether an input sample is likely to be normal or adversarial at runtime by measuring its sensitivity. We evaluated our approach on the MNIST and CIFAR10 dataset. The results show that our approach detects adversarial samples generated by state-of-art attacking methods efficiently and accurately.**

I. Introduction

In recent years, deep neural networks (DNN) have been shown to be useful in a wide range of applications including computer vision [15], speech recognition [53], and malware detection [57]. However, recent research has shown that DNN can be easily fooled [44], [12] by adversarial samples, i.e., normal samples imposed with small, human non-perceptible changes (a.k.a. perturbations). Many DNN-based systems like image classification [30], [33], [6], [51] and speech recognition [7] are shown to be vulnerable to such adversarial samples. This undermines using DNN in safety critical applications like self-driving cars [4] and malware detection [57].

To mitigate the threat of adversarial samples, the machine learning community has proposed multiple approaches to improve the robustness of the DNN model. For example, an intuitive approach is data augmentation. The basic idea is to include adversarial samples into the training data and retrain the DNN [36], [21], [45]. It has been shown that data augmentation improves the DNN to some extent. However, it does not help defend against unseen adversarial samples, especially those obtained through different attacking methods. Alternative approaches include robust optimization and adversarial training [38], [46], [56], [28], which take adversarial perturbation into consideration and solve the robust optimization problem directly during DNN training. However, such approaches usually increase the training cost significantly.

Meanwhile, the software engineering community attempts to tackle the problem using techniques like software testing and verification. In [45], neuron coverage was first proposed to be a criteria for testing DNN. Subsequently, multiple testing metrics based on the range coverage of neurons were proposed [25]. Both white-box testing [35], black-box testing [45] and concolic testing [42] strategies have been proposed to generate adversarial samples for adversarial training. However, testing alone does not help in improving the robustness of DNN, nor does it provide guarantee that a well-tested DNN is robust against new adversarial samples. The alternative approach is to formally verify that a given DNN is robust (or satisfies certain related properties) using techniques like SMT solving [19], [48] and abstract interpretation [11]. However, these techniques usually have non-negligible cost and only work for a limited class of DNN (and properties).

In this work, we provide a complementary perspective and propose an approach for detecting adversarial samples at runtime. The idea is that, given an arbitrary input sample to a DNN, to decide at runtime whether it is likely to be an adversarial sample or not. If it is, we raise an alarm and report that the sample is 'suspicious' with certain confidence. Once detected, it can be rejected or checked depending on different applications. Our detection algorithm integrates mutation testing of DNN models [26] and statistical model checking [2]. It is designed based on the observation that adversarial samples are much more sensitive to mutation on the DNN than normal samples, i.e., if we mutate the DNN slightly, the mutated DNN is more likely to change the label on the adversarial sample than that on the normal one. This is illustrated in Figure 1. The left figure shows a label change on a normal sample, i.e., given a normal sample which is classified as a cat, a label change occurs if the mutated DNN classifies the input as a dog. The right figure shows a label change on an adversarial sample, i.e., given an adversarial sample which is mis-classified as a dog, a label change occurs if the mutated DNN classifies the input as a cat. Our empirical study confirms that the label change rate (LCR) of adversarial samples is significantly higher than that of normal samples against a set of DNN mutants. We thus propose a measure of a sample's sensitivity against DNN mutants in terms of LCR. We further adopt statistical analysis methods like receiver operating characteristic (ROC [8]) to show that we can distinguish adversarial samples and normal samples with high accuracy based on LCR. Our algorithm then takes a DNN model as input, generates a set of DNN mutants,

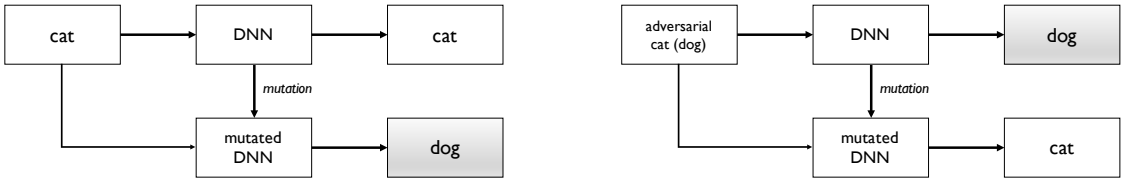

Fig. 1: Label change of a normal sample and an adversarial sample against DNN mutation models.

and applies statistical model checking to check whether the given input sample has a high LCR and thus is likely to be adversarial.

We implement our approach as a self-contained toolkit[1]. We apply our approach on the MNIST and CIFAR10 dataset against the state-of-the-art attacking methods for generating adversarial samples. The results show that our approach detects adversarial samples efficiently with high accuracy. All four DNN mutation operators we experimented with show promising results on detecting 6 groups of adversarial samples, e.g., capable of detecting most of the adversarial samples within around 150 DNN mutants. In particular, using DNN mutants generated by Neuron Activation Inverse (NAI) operator, we manage to detect 96.4% of the adversarial samples with 74.1 mutations for MNIST and 90.6% of the adversarial samples with 86.1 mutations for CIFAR10 on average.

The rest of the paper is organized as follows. Section II provides the necessary background of this work. Section III presents the details of our approach. Section IV shows our experiment results. Section V reviews related works and Section VI concludes.

II. Background

In this section, we first review the necessary background of this work, i.e., state-of-art methods for generating adversarial samples for Deep Neural Networks (DNN), and then define our problem.

A. Adversarial Samples for Deep Neural Networks

Deep learning systems are emerging in many different applications including computer vision, natural language processing, and malware detection. Different DNN architectures like Convolutional Neural networks (CNN) and Recurrent Neural Networks (RNN) have been proposed for different application scenarios. In this work, we focus on DNN classifiers which take a given sample and label the sample accordingly (e.g., as a certain object). In the following, we use x to denote an input sample for a DNN f. We use c_x to denote the ground-truth label of x. Given an input sample x and a DNN f, we can obtain the label of the input x under f by performing forward propagation. x is regarded as an *adversarial sample* with respect to the DNN f if $f(x) \neq c_x$. x is regarded as a *normal sample* with respect to the DNN f if $f(x) = c_x$. Notice that under our definition, those samples in the training/testing dataset wrongly labeled by f are also adversarial samples.

[1]Available online at: removed for anonymity.

Since Szegedy *et al.* made an intriguing discovery that neural networks are vulnerable to adversarial samples in [44], many attacking methods have been developed on how to generate adversarial samples efficiently (e.g., with minimum perturbation). That is, given a normal sample x, an attacker aims to find a minimum perturbation Δx which satisfies $f(x + \boldsymbol{\Delta} x) \neq c_x$. In the following, we briefly introduce several state-of-the-art attacking algorithms.

FGSM: The Fast Gradient Sign Method (FGSM) [13] is designed based on the intuition that we can change the label of an input sample by changing its softmax value to the largest extent, which is represented by its gradient. The implementation of FGSM is straightforward and efficient. By simply adding up the sign of gradient of the cost function with respect to the input, we could quickly obtain a potential adversarial counterpart of a normal sample by the follow formulation:

$$\hat{x} = x + \epsilon \mathbf{sign}(\nabla \mathbf{J}(\theta, x, c_x))$$

, where $\mathbf{J}$ is the cost used to train the model and θ is the parameters. Notice that FGSM does not guarantee that the adversarial perturbation is minimal.

JSMA: Jacobian-based Saliency Map Attack(JSMA) [34] is devised to attack one model with minimal perturbations and enable the adversarial samples to mislead the target model into classifying it with certain (attacker-desired) label. It is a greedy algorithm that changes one pixel during each iteration to increase the probability of having the target label. Based on the Jacobian matrix, the idea is to calculate a saliency map to model the impact that each pixel imposes on the target classification. With the saliency map, the algorithm picks the pixel which may have the most significant influence on the desired change and then increases it to the maximum value. The process is repeated until it reaches one of the stopping criteria, i.e., the number of pixels modified has reached the bound, or the target label has been achieved. Define

$$\begin{cases} a_i = \dfrac{\partial \mathbf{F_t}(x)}{\partial X_i} \\ b_i = \displaystyle\sum_{k \neq t} \dfrac{\partial \mathbf{F_k}(x)}{\partial X_i} \end{cases}$$

Then, the saliency map at each iteration is defined as follow:

$$S(x,t)_i = \begin{cases} a_i \times |b_i| & if\ a_i > 0\ and\ b_i < 0 \\ 0 & otherwise \end{cases}$$

However, it is too strict to select one pixel at a time because few pixels could meet that definition. Thus, instead of picking one pixel at a time, the authors proposed to pick two pixels to modify according to the follow objective:

$$\arg\max_{(p_1,p_2)} \left(\frac{\partial \mathbf{F}_t(x)}{\partial x_{p_1}} + \frac{\partial \mathbf{F}_t(x)}{\partial x_{p_2}} \right) \times \left| \sum_{i=p_1,p_2} \sum_{k \neq t} \frac{\partial \mathbf{F}_k(x)}{\partial x_i} \right|$$

where(p_1, p_2) is the candidate pair, and t is the target class.

JSMA is relatively time-consuming and memory-consuming since it needs to compute the Jacobian matrix and pick out a pair from nearly $\binom{n}{2}$ candidate pairs at each iteration.

DeepFool: The idea of DeepFool (DF) is to make the normal samples cross the decision boundary with minimum perturbations [30]. The authors first deduced an iterative algorithm for binary classifiers with Tayler's Formula, and then analytically derived the solution for multi-class classifiers. The exact derivation process is complicated and thus we refer the readers to [30] for details.

C&W: Carlini *et al.* [6] proposed a group of attacks based on three distance metrics. The key idea is to solve an optimization problem which minimizes the perturbation imposed on the normal sample (with certain distance metric) and maximizes the probability of the target class label. The objective function is as follow:

$$\arg\min \Delta x + c \cdot f(\hat{x}, t)$$

where Δx is defined according to some distance metric, e.g, L_0, L_2, L_∞, $\hat{x} = x + \Delta x$ is the clipped adversarial sample and t is its target label. The idea is to devise a clip function for the adversarial sample such that the value of each pixel dose not exceed the legal range. The clip function and the best loss function according to [6] are shown as follows.

$$\begin{aligned} clip: \hat{x} &= 0.5(tanh(\tilde{x}) + 1) \\ loss: f(\hat{x}, t) &= \max(\max\{G(\hat{x})_c : c \neq t\} - G(\hat{x})_t, 0) \end{aligned}$$

where $G(x)$ denotes the output vector of a model and t is the target class. Readers can refer to [6] for details.

Black-Box: All the above mentioned attacks are white-box attacks which means that the attackers require the full knowledge of the DNN model. Black-Box (BB) attack only needs to know the output of the DNN model given a certain input sample. The idea is to train a substitute model to mimic the behaviors of the targeted model with data augmentation. Then, it applies one of the existing attack algorithm, e.g., FGSM and JSMA, to generate adversarial samples for the substitute model. The key assumption to its success is that the adversarial samples transfer between different model architectures [44], [13].

B. Problem Definition

Observing that adversarial samples are relatively easy to craft, a variety of defense mechanisms against adversarial samples have been proposed [14], [28], [52], [27], [39], as we have briefly introduced in Section I. However, Athalye *et al.* [1] systematically evaluated the state-of-the-art defense mechanisms recently and showed that most of them are ineffective against the state-of-the-art attacking methods. Alternative defense mechanisms are thus desirable.

In this work, we take a complementary perspective and propose to detect adversarial samples at runtime borrowing techniques from the software engineering community. *The problem is: given an input sample x to a deployed DNN f, how can we efficiently and accurately decide whether $f(x) = c_x$ (i.e., a normal sample) or not (i.e., an adversarial sample)?* If we know that x is likely an adversarial sample, we could reject or further check it. Furthermore, can we quantify our confidence on the drawn conclusion?

III. Our approach

Our approach is based on the hypothesis that, in most cases adversarial samples are more 'sensitive' to mutations on the DNN model than normal samples. That is, if we generate a set of slightly mutated DNN models based on the given DNN model, the mutated DNN models are more likely to label an adversarial sample with a label different from the label generated by the original DNN model, as illustrated in Figure 1. In other words, our approach is designed based on a measure of sensitivity for differentiating adversarial samples and normal samples. In the literature, multiple measures have been proposed to capture their differences, e.g., density estimate, model uncertainty estimate [9], and sensitivity to input perturbation [47]. Our measure however allows us to detect adversarial samples at runtime efficiently through model mutation testing.

A. Mutating Deep Neural Networks

In order to test our hypothesis (and develop a practical algorithm), we need a systematic way of generating mutants of a given DNN model. We adopt the method developed in [26], which is a proposal of applying mutation testing to DNN, and thus is different from our work. Mutation testing [18] is a well-known technique to evaluate the quality of a test suite. The idea is to generate multiple mutations of the program under test, by applying a set of mutation operators, in order to see how many of the mutants can be killed by the test suite. The definition of the mutation operators is a core component of the technique. Given the difference between traditional software systems and DNN, mutation operators designed for traditional programs cannot be directly applied to DNN. In [26], Ma *et al.* introduced a set of mutation operators for DNN-based systems at different levels like source level (e.g., the training data and training programs) and model level (e.g., the DNN model).

In this work, we require a large group of slightly mutated models for runtime adversarial sample detection. Of all the mutation operators proposed in [26], mutation operators defined at the source level are not considered. The reason is that we would need to train the mutated models from scratch which is often time-consuming. We thus focus on the model-level operators, which modify the original model directly to

TABLE I: DNN model mutation operators

Mutation Operator	Level	Description
Gaussian Fuzzing (GF)	Weight	Fuzz weight by Gaussian Distribution
Weight Shuffling (WS)	Neuron	Shuffle selected weights
Neuron Switch (NS)	Neuron	Switch two neurons within a layer
Neuron Activation Inverse (NAI)	Neuron	Change the activation status of a neuron

obtain mutated models without training. Specifically, we adopt four of the eight defined operators from [26] shown in Table I. For example, NAI means that we change the activation status of a certain number of neurons in the original model. Notice that the other four operators defined in [26] are not applicable due to the specific architecture of the deep learning models we focus on in this work.

B. Evaluating Our Hypothesis

We first conduct experiments to measure the label change rate (LCR) of adversarial samples and normal samples when we feed them into a set of mutated DNN models. Given an input sample x (either normal or adversarial) and a DNN model f, we first adopt the model mutation operators shown in Table I to obtain a set of mutated models. Note that some of the resultant mutated models may be of low quality, i.e., their classification accuracy on the test data drops significantly. We discharge those low quality ones and only keep those accurate mutated models which retain an accuracy on the test data, i.e., at least 90% of the accuracy of the original model. Once we obtain such a set of mutated models F, we then obtain the label $f_i(x)$ of the input sample x on every mutated model $f_i \in F$. We define LCR on a sample x as follows (with respect to F).

$$\varsigma(x) = \frac{|\{f_i | f_i \in F \ and \ f_i(x) \neq f(x)\}|}{|F|}$$

, where $|S|$ is the number of elements in a set S. Intuitively, $\varsigma(x)$ measures how sensitive an input sample x is on the mutation of DNN models.

Table II summarizes our empirical study on measuring $\varsigma(x)$ using two popular dataset, i.e., the MNIST and CIFAR10 dataset, and multiple state-of-the-art attacking methods. A total of 500 mutated models are generated using NAI operator which randomly selects some neurons and change its activation status. The first column shows the name of the dataset. The second shows the mutation rate, i.e., the percentage of the neurons whose activation status are changed. The third shows the average LCR (with confidence interval of 99% significance level) of 1000 normal samples randomly selected from the testing set. The remaining columns show the average LCR (with confidence interval of 99% significance level) of 1000 adversarial samples which are generated using state-of-the-art methods. Note that column 'Wrongly Labeled' are samples from the testing set which are wrongly labeled by the original DNN model.

Based on the results, we can observe that at any mutation rate, the ς values of the adversarial samples are significantly higher than those of the normal samples.

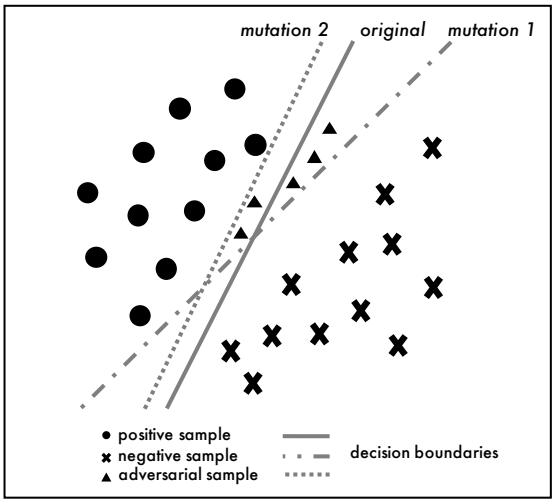

Fig. 2: Our explanatory model of the mutation testing effect.

> ς_{adv} *is significantly larger than* ς_{nor}.

Further study on the LCR distance between normal and adversarial samples with respect to different model mutation operators is presented in Section IV. The results are consistent. A practical implication of the observation is that given an input sample x, we could potentially detect whether x is likely to be normal or adversarial by checking $\varsigma(x)$.

C. Explanatory Model

In the following, we use a simple model to explain the above observation. Recall that adversarial samples are generated in a way which tries to minimize the modification to a normal sample while is still able to cross the decision boundary. Different kinds of attacks use different approaches to achieve this goal. Our hypothesis is that most adversarial samples generated by existing methods are near the decision boundary (to minimize the modification). As a result, as we randomly mutate the model and perturb the decision boundary, adversarial samples are more likely to cross the mutated decision boundaries, i.e., if we feed an adversarial sample to a mutated model, the output label has a higher chance to change from its original label. This is illustrated visually in Figure 2.

D. The Detection Algorithm

The results shown in Table II suggests that we can use LCR to distinguish adversarial samples and normal samples. In the following, we present an algorithm which is designed to detect adversarial samples at runtime based on measuring the LCR of a given sample. The algorithm is based on the idea of statistical model checking [23].

The inputs of our algorithm are a DNN model f, a sample x and a threshold ς_h which is used to decide whether the input is adversarial. We will discuss later on how to identify ς_h systematically. The basic idea of our algorithm is to use

TABLE II: Average ς (shown in percentage with confidence interval of 99% significance level) for normal samples and adversarial samples under 500 NAI mutated models.

Dataset	Mutation rate	Normal samples	Adversarial samples					
			Wrong labeled	FGSM	JSMA	C&W	Black-Box	Deepfool
MNIST	0.01	1.28 ± 0.24	14.58 ± 2.64	47.56 ± 3.56	50.80 ± 2.46	12.07 ± 1.26	44.94 ± 3.43	37.62 ± 2.83
	0.03	3.06 ± 0.44	27.16 ± 3.11	52.12 ± 3.04	57.86 ± 2.02	21.88 ± 1.38	51.15 ± 2.91	46.61 ± 2.43
	0.05	3.88 ± 0.53	32.53 ± 3.15	54.54 ± 2.80	59.07 ± 1.95	27.73 ± 1.37	53.97 ± 2.67	50.30 ± 2.24
CIFAR10	0.001	2.20 ± 0.55	17.95 ± 1.39	14.06 ± 1.33	28.65 ± 1.30	19.77 ± 1.41	10.36 ± 1.06	30.84 ± 1.37
	0.003	5.05 ± 0.91	32.18 ± 1.62	27.87 ± 1.71	47.75 ± 1.27	33.95 ± 1.60	21.66 ± 1.38	47.70 ± 1.23
	0.005	7.28 ± 1.12	39.76 ± 1.70	36.19 ± 1.81	56.02 ± 1.29	41.22 ± 1.64	27.57 ± 1.5	54.41 ± 1.21

acceptance sampling to decide the truthfulness of two mutual exclusive hypothesis.

$$H_0 : \quad \varsigma(x) > \varsigma_h$$
$$H_1 : \quad \varsigma(x) \leq \varsigma_h$$

Three (standard) additional parameters, α, β and δ, are used to control the probability of making an error. That is, we would like to guarantee that the probability of a Type-I (respectively, a Type-II) error, which rejects H_0 (respectively, H_1) while H_0 (respectively, H_1) holds, is less or equal to α (respectively, β). The test needs to be relaxed with a confidence interval $(r-\delta, r+\delta)$, where neither hypothesis is rejected and the test continues to bound both types of errors [23]. In practice, the parameters (i.e., (α, β), and δ) can often be decided by how much testing resources are available. In general, more resource is required for a smaller error bound.

Our detection algorithm keeps generating accurate mutated models (with an accuracy more than certain threshold on the testing data) from the original model and evaluating $\varsigma(x)$ until a stopping condition is satisfied. We remark that we could generate a set of accurate mutated models before-hand and simply use them at runtime to further save detection time.

There are two main methods to decide when the testing process can be stopped, i.e., we have sufficient confidence to reject a hypothesis. One is the fixed-size sampling test (FSST), which requires running a predefined number of tests. One difficulty of this approach is to find an appropriate number of tests to be performed such that the error bounds are valid [23]. The other approach is the sequential probability ratio test (SPRT [23]). SPRT dynamically decides whether to reject or not a hypothesis every time after we update $\varsigma(x)$, which requires a variable number of mutated models. SPRT is usually faster than FSST as the testing process ends as soon as a conclusion is made [23].

In this work, we use SPRT for the detection. The details of our SPRT-based algorithm is shown in Algorithm 1. The inputs of the detection algorithm include the input sample x, the original DNN model f, a mutation rate γ, and a threshold of LCR ς_h. Besides, the detection is error bounded by $\langle \alpha, \beta \rangle$ and relaxed with an indifference region σ. To apply SPRT, we keep generating accurate mutated models at line 5. The details of generating mutated models using the four operators in Table I are shown in Algorithm 2, Algorithm 3, Algorithm 4, and Algorithm 5 respectively. We then evaluate whether $f_i(x) = f(x)$ at line 7. If we observe a label change of x using the mutated model f_i, we calculate and update the SPRT probability ratio at line 9 as:

$$pr = \frac{p_1^z(1-p_1)^{n-z}}{p_0^z(1-p_0)^{n-z}}$$

, with $p_1 = \varsigma_h + \sigma$ and $p_0 = \varsigma_h - \sigma$. The algorithm stops whenever a hypothesis is accepted either at line 11 or line 14. We remark that SPRT is guaranteed to terminate with probability 1 [2].

We briefly introduce the NAI operator shown in Algorithm 2 as an example of the four mutation operators. We first obtain the set of N unique neurons[2] at line 1. Then we randomly select $\lceil N \times \gamma \rceil$ neurons (γ is the mutation rate) for activation status inverse at line 2. Afterwards, we traverse the model f layer by layer at line 3 and take those selected neurons at line 4. We then inverse the activation status of the selected neurons by multiplying their weights with -1 at line 7.

IV. Implementation and Evaluation

We have implemented our approach in a self-contained toolkit which is available online[3]. It is implemented in Python with a total of 7.5k lines of code. In the following, we evaluate the effectiveness and efficiency of our approach through multiple experiments.

A. Experiment Settings

a) Datasets and Models: We adopt two popular image datasets for our evaluation: MNIST and CIFAR10. Each dataset has 60000 images for training and 10000 images for testing. The target models for MNIST and CIFAR10 are LeNet [22] and GooglLeNet [43] respectively. The accuracy of our trained models on training and testing dataset are 98.5%/98.3% for MNIST and 99.7%/90.5% for CIFAR10 respectively, which both achieve state-of-the-art performance.

b) Mutated models generation: We employ the four mutation operators shown in Table I to generate mutated models. In total, we have 236 neurons for the MNIST model and 7914 neurons for the CIFAR10 model. For each mutation operator, we generate three groups of mutation models from

[2]For convolutional layer, each slide of convolutional kernel is regarded as a neuron

[3]removed for anonymity

Algorithm 1: SPRT-Detect$(x, f, \gamma, \varsigma_h, \alpha, \beta, \sigma)$

Let $stop = false$;
Let $z = 0$ be the number of mutated models f_i that satisfy $f_i(x) \neq f(x)$;
Let $n = 0$ be the total number of generated mutated models so far;
while *!stop* **do**
  Apply a mutation operator to randomly generate an accurate mutation model f_i of f with mutation rate γ;
  $n = n + 1$;
  if $f_i(x) \neq f(x)$ **then**
    $z = z + 1$;
    Calculate the SPRT probability ratio as pr;
    if $pr \geq \frac{1-\beta}{\alpha}$ **then**
      Accept the hypothesis that $\varsigma(x) > \varsigma_h$ and report the input as an adversarial sample with error bounded by β;
      return;
    if $pr \leq \frac{\beta}{1-\alpha}$ **then**
      Accept the hypothesis that $\varsigma(x) \leq \varsigma_h$ and report the input as a normal sample with error bounded by α;
      return;

Algorithm 2: $NAI(f, \gamma)$

Let N be the set of unique neurons;
Randomly select $\lceil N \times \gamma \rceil$ unique neurons;
for *every layer in* f **do**
  Let Q be the set of selected neurons in this layer;
  if $Q \neq \emptyset$ **then**
    for $q \leftarrow Q$ **do**
      $q.weight = -1 \cdot q.weight$;

Algorithm 3: $GF(f, \gamma)$

Let W be the parameters of f;
Extract the parameters of f layer by layer;
Let N be the total number of parameters of f;
Randomly select $\lceil N \times \gamma \rceil$ parameters to fuzz;
for *every layer in* f **do**
  Let $W[i]$ be the parameters of this layer;
  Find all the selected parameters P in $W[i]$;
  if $P \neq \emptyset$ **then**
    Let $\mu = Avg(W[i])$;
    Let $\sigma = Std(W[i])$;
    for *every parameter in* P **do**
      Randomly assign the parameter according to $\mathcal{N}(\mu, \sigma^2)$;

Algorithm 4: $WS(f, \gamma)$

Let N be the set of unique neurons;
Randomly select $\lceil N \times \gamma \rceil$ unique neurons to shuffle ;
for *every layer in* f **do**
  Let Q be the set of selected neurons in this layer;
  if $Q \neq \emptyset$ **then**
    for $q \leftarrow Q$ **do**
      $q.weight = \text{Shuffle}(q.weight)$;

the original trained model using different mutation rate to see its effect. The mutation rate we use for the MNIST model is $\{0.01, 0.03, 0.05\}$ and $\{0.001, 0.003, 0.005\}$ for the CIFAR10 model (since there are more neurons). Note that some mutation models may have significantly worse performance, so not all mutated models are valid. In our experiment, we only keep those mutation models whose accuracy on the testing dataset is not lower than 90% of that of its seed model. For each mutation rate, we generate 500 such accurate mutated models for our detection algorithm.

c) Adversarial samples generation: We test our detection algorithm against four state-of-the-art attacks in Clverhans [31] and Deepfool [30] (detailed in Section II). For each kind of attack, we generate a set of adversarial samples for evaluation. The parameters for each kind of attack to generate the adversarial samples are summarized as follows.

- **FGSM**: There is only one parameter to control the scale of perturbation. We set it as 0.35 for MNIST and 0.03 for CIAFR10 according the original paper.
- **JSMA**: There is only one parameter to control the maximum distortion. We set it as 12% for both datasets, which is slightly smaller than the original paper.
- **C&W**: There are three types of attacks proposed in [6]: L_0, L_2 and L_∞. We adopt L_2 attack according to the author's recommendation. We also set the scale coefficient to be 0.6 for both datasets. We set the iteration number to be 10000 for MNIST and 1000 for CIFAR10 according to the original paper.
- **Deepfool**: We set the maximum number of iterations to be 50 and the termination criterion (to prevent vanishing updates) to be 0.02 for both datasets, which is a default setting in the original paper.
- **Black-Box**: The key setting of the Black-Box attack is to train a substitute model of the target model. The substitute model for MNIST is the first model defined in Appedix A of [32]. For CIFAR10, we use the LeNet [22] as the surrogate model. The attack algorithm we used in Black-Box is FGSM.

For each attack, we make 1000 attempts to generate adversarial samples. Notice that not all attempts are successful and as a result we manage to generate no more than 1000 adversarial samples for each attack. Further recall that according

Algorithm 5: $NS(f, \gamma)$

for *every layer in* f **do**
  Let N be the number of unique neurons in this layer;
  Randomly select $\lceil N \times \gamma \rceil$ unique neurons;
  Let Q be the set of selected neurons;
  Randomly switch the weights of neurons in Q;

TABLE III: Number of samples in each group.

Dataset	Attack	Samples
MNIST	Normal	1000
	Wrongly-labeled	171
	FGSM	1000
	JSMA	1000
	BB	1000
	C&W	743
	Deepfool	1000
CIFAR10	Normal	1000
	Wrongly-labeled	951
	FGSM	1000
	JSMA	1000
	BB	1000
	C&W	1000
	Deepfool	1000

to our definition, those samples in the testing dataset which are wrongly labeled by the trained DNN are also adversarial samples. Thus, in addition to the adversarial samples generated from the attacking methods, we attempt to randomly select 1000 samples from the testing dataset which are wrongly classified by the target model. Table III summarizes the number of normal samples and valid adversarial samples for each kind of attack used for the experiments.

B. Evaluation Metrics

a) Distance of label change rate: We use $d_{lcr} = \varsigma_{adv}/\varsigma_{nor}$ where ς_{adv} (and ς_{nor}) is the average LCR of adversarial samples (and normal samples) to measure the distance between the LCR of adversarial samples and normal samples. The larger the value is, the more significant is the difference.

b) Receiver characteristics operator: Since our detection algorithm works based on a threshold LCR ς_h, we first adopt receiver characteristic operator (ROC) curve to see how good our proposed feature, i.e., LCR under model mutation, is to distinguish adversarial and normal samples [8], [9]. The ROC curve plots the true positive rate (tpr) against false positive rate (fpr) for every possible threshold for the classification. From the ROC curve, we could further calculate the area under the ROC curve (AUROC) to characterize how well the feature performs. A perfect classifier (when all the possible thresholds yield true positive rate 1 and false positive rate 0 for distinguishing normal and adversarial samples) will have AUROC 1. The closer is AUROC to 1, the better is the feature.

c) Accuracy of detection: The accuracy of the detection is defined in a standard way as follows. Given a set of images X (labeled as normal or adversarial), what is the percentage that our detector correctly classifies it as normal or adversarial? Notice that the accuracy of detecting adversarial samples is equivalent to tpr and the accuracy of detecting normal samples is equivalent to $1 - fpr$. The higher the accuracy, the better is our detection algorithm.

C. Research Questions

RQ1: Is there a significant difference between the LCR of adversarial samples and normal samples under different model mutations? To answer the question, we calculate the average LCR of the set of normal samples and the set of adversarial samples generated as described above with a set of mutated models using different mutation operators. A set of 500 mutants are generated for each mutation operator (note that mutation rate 0.001 is too low for NS to generate mutated models for CIFAR10 model and thus omitted). According to the detailed results summarized in Tabel II and IV, we have the following answer.

> *Answer to RQ1: Adversarial samples have significantly higher LCR under model mutation testing than normal samples.*

In addition, we have the following observations.

- Adversarial samples generated from every kind of attack have significantly larger LCR than normal samples under mutation models under any mutation rate, and different kind of attack have different LCR. We can see that the LCR of normal samples are very low (i.e., comparable to the testing error) and that of adversarial samples are much higher. Figure 3 shows the distance between LCR of adversarial samples and normal samples for different mutation operators. We can see that the distance is mostly larger than 10 and can be up to 375, which well supports our answer to RQ1. We can also observe that adversarial samples generated by FGSM/JSMA/Deepfool/Black-box have relatively higher LCR distance than those generated by CW and those wrong-labeled samples in the original dataset. In general, our detection algorithm is able to detect attacks with larger distance faster and better.
- As we increase the model mutation rate, the LCR of both normal samples and adversarial samples increases (which is expected) and the distance between them decreases. We can observe from Table IV that the LCR increases with an increasing model mutation rate in all cases. From Figure 3, we see that a smaller model mutation rate like 0.01 for MNIST and 0.001 for CIFAR10 have the largest LCR distance. This is probably because as we increase the mutation rate, normal samples are more sensitive in terms of the change of LCR since it is a much smaller number.
- Like adversarial samples generated by different attacking methods, wrongly labeled samples also have significantly larger LCR than normal samples. This suggests that wrongly labeled samples are also sensitive to the change

TABLE IV: Label change rate (confidence interval with 99% significance level) for each group of samples under model mutation testing with different mutation operators (NAI result is shown previously in Table II). The results are shown in percentage.

Mutation operator	Dataset	Mutation rate	Normal samples	Adversarial samples					
				Wrong labeled	FGSM	JSMA	C&W	Black-Box	Deepfool
NS	MNIST	0.01	0.12 ± 0.07	3.78 ± 0.94	44.67 ± 3.92	36.03 ± 3.24	3.42 ± 0.79	40.06 ± 3.82	26.09 ± 3.16
		0.03	0.37 ± 0.19	10.78 ± 2.30	46.32 ± 3.71	47.45 ± 2.61	8.93 ± 1.16	43.05 ± 3.59	34.20 ± 2.92
		0.05	0.89 ± 0.35	19.30 ± 3.18	48.91 ± 3.41	56.51 ± 2.11	15.87 ± 1.53	46.94 ± 3.29	42.69 ± 2.65
	CIFAR10	0.001	-	-	-	-	-	-	-
		0.003	0.02 ± 0.03	0.3 ± 0.15	0.3 ± 0.16	0.46 ± 0.16	0.37 ± 0.18	0.08 ± 0.05	0.86 ± 0.24
		0.005	0.94 ± 0.4	10.12 ± 1.19	7.16 ± 1.06	16.07 ± 1.21	11.04 ± 1.19	4.61 ± 0.8	19.05 ± 1.37
WS	MNIST	0.01	0.93 ± 0.18	9.83 ± 2.33	46.04 ± 3.73	46.96 ± 2.67	7.98 ± 1.15	42.42 ± 3.62	33.41 ± 2.97
		0.03	3.03 ± 0.35	21.84 ± 3.11	49.83 ± 3.26	56.01 ± 2.10	17.01 ± 1.38	47.98 ± 3.14	43.07 ± 2.60
		0.05	3.83 ± 0.42	26.96 ± 3.26	51.46 ± 3.06	57.56 ± 2.00	21.03 ± 1.40	50.20 ± 2.94	46.37 ± 2.46
	CIFAR10	0.001	0.79 ± 0.35	9.04 ± 1.17	6.43 ± 1.05	14.85 ± 1.27	10.01 ± 1.18	9.11 ± 0.74	18.78 ± 1.46
		0.003	2.01 ± 0.55	17.0 ± 1.53	12.88 ± 0.145	29.42 ± 1.55	18.42 ± 1.55	8.49 ± 1.06	32.63 ± 1.63
		0.005	2.69 ± 0.65	21.6 ± 1.67	17.21 ± 1.67	37.69 ± 1.63	23.40 ± 1.69	11.15 ± 1.22	40.03 ± 1.63
GF	MNIST	0.01	0.57 ± 0.30	16.75 ± 3.33	47.87 ± 3.54	56.39 ± 2.14	14.27 ± 1.56	45.56 ± 3.41	41.07 ± 2.76
		0.03	1.39 ± 0.46	27.00 ± 3.40	51.87 ± 3.10	60.64 ± 1.85	22.10 ± 1.64	50.59 ± 2.97	48.06 ± 2.41
		0.05	2.49 ± 0.59	33.28 ± 3.28	55.02 ± 2.77	62.36 ± 1.74	25.87 ± 1.55	53.38 ± 2.68	51.60 ± 2.19
	CIFAR10	0.001	1.42 ± 0.51	15.36 ± 1.52	11.42 ± 1.42	26.52 ± 1.53	17.0 ± 1.51	8.05 ± 1.10	31.36 ± 1.68
		0.003	2.89 ± 0.75	25.31 ± 1.75	20.71 ± 1.79	41.69 ± 1.54	26.59 ± 1.75	13.75 ± 1.34	45.8 ± 1.57
		0.005	4.09 ± 0.91	31.97 ± 1.86	27.69 ± 1.97	50.07 ± 1.52	32.94 ± 1.82	18.29 ± 1.48	53.67 ± 1.51

of decision boundaries from model mutations as adversarial samples. They are the same as the adversarial samples from a statistical point of view and thus can be potentially detected.

RQ2: How good is the LCR under model mutation as an indicator for the detection of adversarial samples? To answer the question, we further investigate the ROC curve using LCR as the indicator of classifying an input sample as normal or adversarial. We compare our proposed feature, i.e., LCR under model mutations with two current baseline approaches. The first baseline (referred as baseline 1) is a combination of density estimate and model uncertainty estimate as joint features [9]. The second baseline (referred as baseline 2) is based on the LCR of imposing random perturbations on the input sample [47].

Table V presents the AUROC results under different model mutation operators. We compare our results with two baselines introduced above. The best AUROC results among the three approaches are in bold. We could observe that our proposed feature beats both baselines in over half the cases (excluding Deepfool which we do not have any reported baseline results), while baseline 1 and baseline 2 only win 1 and 3 cases respectively. We could also see that the AUROC results are mostly very close to 1 (a perfect classifier), i.e., usually larger than 0.9, which suggests that we could achieve high accuracy using the proposed feature to distinguish adversarial samples. We thus have the following answer to RQ2.

Answer to RQ2: LCR under model mutations could outperform current baselines to detect adversarial samples.

RQ3: How effective is our detection algorithm based on LCR under model mutation? To answer the question, we apply our detection algorithm (Algorithm 1) on each set of adversarial samples generated using each attack and evaluate the accuracy of the detection in Figure 4. We also report the accuracy of our

TABLE V: AUROC results. BL means 'baseline'.

Dataset	Attack	BL 1	BL 2	NAI	GF	NS	WS
MNIST	FGSM	0.9057	**0.9767**	0.9744	0.9747	0.9554	0.9648
	JSMA	0.9813	0.9946	0.9965	**0.9975**	0.9975	0.9969
	CW	**0.9794**	0.9394	0.9576	0.9521	0.909	0.9225
	BB	–	0.9403	**0.9789**	0.9763	0.9631	0.9725
	DF	–	–	0.9881	**0.9889**	0.9853	0.9864
	WL	–	0.9696	0.9689	**0.9727**	0.9612	0.9692
CIFAR10	FGSM	0.7223	**0.9099**	0.8956	0.8779	0.7559	0.8458
	JSMA	0.9152	0.8753	0.9733	**0.9737**	0.9355	0.9729
	CW	0.9217	0.8385	**0.926**	0.9205	0.8464	0.8994
	BB	–	**0.9251**	0.874	0.8371	0.7068	0.8702
	DF	–	–	0.974	**0.9786**	0.9549	0.9753
	WL	–	0.9148	**0.9185**	0.9146	0.8331	0.876

algorithm on a set of normal samples. The results are based on the set of models generated using mutation rate 0.05 for MNIST and 0.005 for CIFAR10 as they have good balance between detecting adversarial and normal samples.

We set the parameters of Algorithm 1 as follows. Since different kind of attacks have different LCR but the LCR of normal sample is relatively stable, we choose to test against the LCR of normal samples. Specifically, we set the threshold ς_h to be $\rho \cdot \varsigma_{nr}$, where ς_{nr} is the upper bound of the confidence interval of ς_{nor} and ρ is a hyper parameter to control the sensitivity of our detection algorithm. The smaller ρ is, the more sensitive our algorithm is to detect adversarial samples. The error bounds for SPRT is set as $\alpha = 0.05, \beta = 0.05$. The indifference region is set as $0.1 \cdot \varsigma_{nr}$.

Figure 4 shows the detection accuracy and average number of model mutants needed for the detection using the 4 mutation operators for MNIST and CIFAR10 dataset respectively. We could observe that our detection algorithm achieves high accuracy on every kind of attack for every mutation operator. On average, the GF/NAI/NS/WS operators achieves accuracy of 94.9%/96.4%/83.9%/91.4% with 75.5/74.1/145.3/105.4 mutated models for MNIST (with ρ=1) and 85.5%/90.6%/56.6%/74.8% (with ρ=1) with

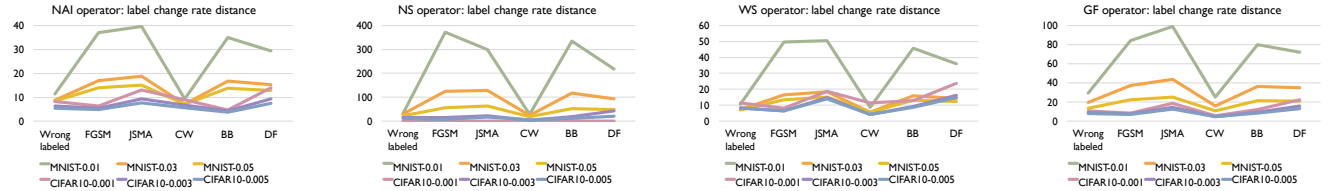

Fig. 3: LCR distance between normal samples and adversarial samples using different mutation operators

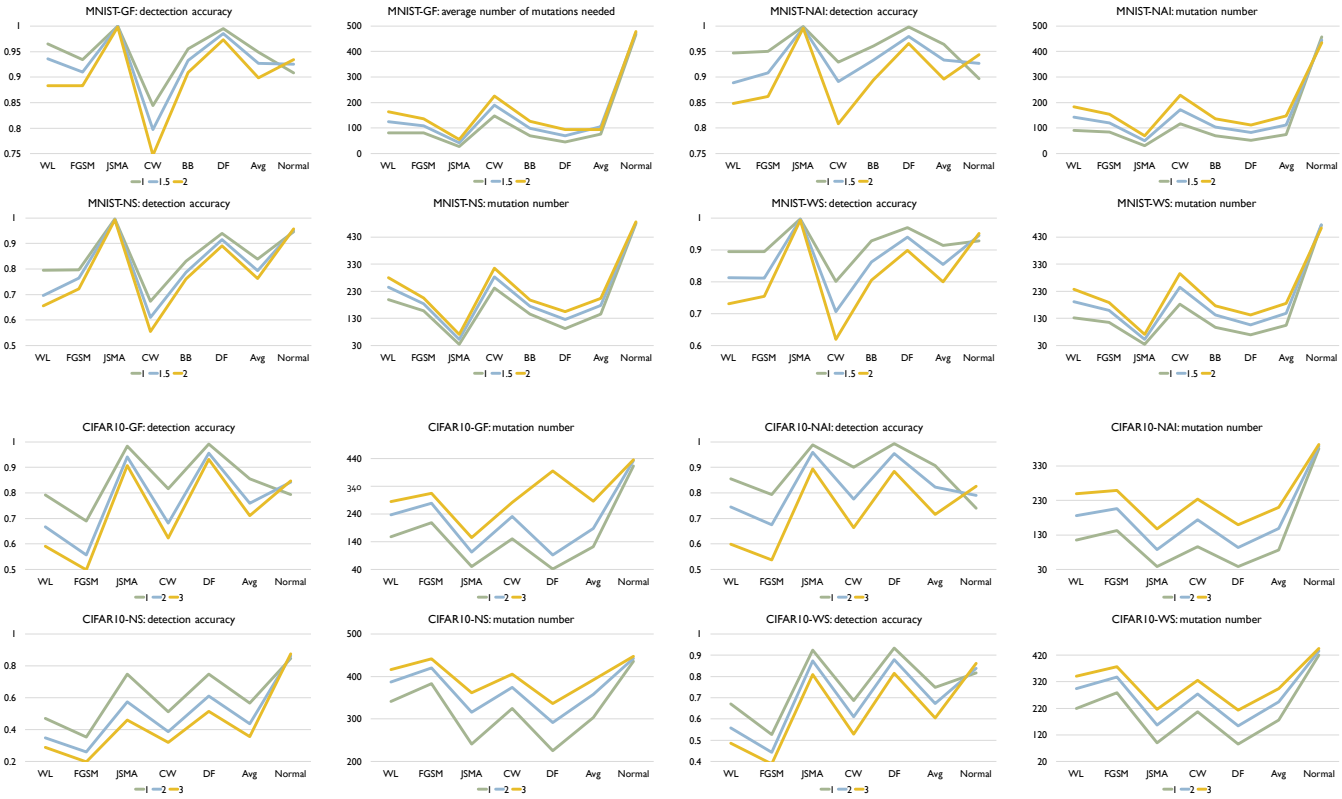

Fig. 4: Detection accuracy and number of mutated models needed.

121.7/86.1/303/176.2 mutated models for CIFAR10 on detecting the 6 kinds of adversarial samples. Meanwhile, we maintain high detection accuracy of normal samples as well, i.e., 90.8%/89.7%/94.7%/92.9% for MNIST (with ρ=1) and 79.3%/74%/84.6%/81.6% (with ρ=1) for CIFAR10 for the above 4 operators respectively. Notice that for CIFAR10, we could not train a good substitute model (the accuracy is below 50%) using Black-box attack and thus have no result. The results show that our detection algorithm is able to detect most of adversarial samples effectively. In addition, we observe that the more accurate is the original (and as a result the mutated) DNN model is (e.g., MNIST), the better is our algorithm. Besides, we are able to achieve accuracy close to 1 for JSMA and DF. We also recommend to use NAI/GF operators over NS/WS operators as they have consistently better performance than the others. We thus have the following answer to RQ3.

Answer to RQ3: Our detection algorithm based on statistical model checking could effectively detect adversarial samples.

Effect of ρ In this experiment, we vary the hyper parameter ρ to see its effect on the detection. As shown in Figure 4, we set ρ as $\{1, 1.5, 2\}$ for MNIST and $\{1, 2, 3\}$ for CIFAR10. We could observe that as we increase ρ, we have a lower accuracy on detecting adversarial samples but a higher accuracy on detecting normal samples. The reason is that as we increase ρ, the threshold for the detection increases. In this case, our algorithm will be less sensitive to detect adversarial samples since the threshold is higher. We could also observe that we would need more mutations with a higher threshold. In summary, the selection of ρ could be application specific and our practical guide is to set a small ρ if the application has a high safety requirement and vice versa.

TABLE VI: Cost analysis of our algorithm.

Dataset	c_f	operator	c_g	n
MNIST	0.7 ms	NAI	6.191 s	68.7789
	0.5 ms	NS	6.336 s	173.0040
	0.3 ms	WS	7.657 s	107.6702
	0.3 ms	GF	1.398 s	91.1747
CIFAR10	0.3 ms	NAI	16.101 s	69.0873
	0.5 ms	NS	9.475 s	283.9628
	0.4 ms	WS	9.251 s	165.6373
	0.4 ms	GF	11.894 s	127.2767

RQ4: What is the cost of our detection algorithm? The cost of our algorithm mainly consists of two parts, i.e., generating mutated models (denoted by c_g) and performing forward propagation (denoted by c_f) to obtain the label of an input sample by a DNN model. The total cost of detecting an input sample is thus $C = n \cdot (c_g + c_f)$, where n is the number of mutants needed to draw a conclusion based on Algorithm 1.

We estimate c_f by performing forward propagation for 10000 images on a MNIST and CIFAR10 model respectively. The detailed results are shown in Tabel VI. Note that c_g is the time used to generate an accurate model (retaining at least 90% accuracy of the original model) and the cost to generate an arbitrary mutated model is much less. In practice, we could generate and cache a set of mutated models for the detection of a set of samples. Given a set of m samples, the total cost for the detection is reduced to $C(m) = m{\cdot}n{\cdot}c_f{+}n{*}c_g$. In practice, our algorithm could detect an input sample within 0.1 second (with cached models) using a single machine. We remark that our algorithm can be parallelized easily by evaluating a set of models at the same time which would reduce the cost significantly. We thus have the following answer to RQ4.

Answer to RQ4: Our detection algorithm is lightweight and easy to parallel.

D. Threats to Validity

First, our experiment is based on a limited set of test subjects so far. Our experience is that the more accurate the original model and the mutated models are, the more effective and more efficient our detection algorithm is. The reason is that the LCR distance between adversarial samples and normal samples will be larger if the model is more accurate, which is good for our detection. In some applications, however, the accuracy of the original models may not be high. Secondly, the detection algorithm will have some false positives. Since our detection algorithm is threshold-based, there will be some false alarms along with the detection. Meanwhile, there is a tradeoff between avoiding false positives or false negatives as discussed above (i.e., in the selection of ρ). Thirdly, the detection of normal samples typically needs more mutations. The reason is that we choose to test against ς_{nor} since we do not know ς_{adv} for an unknown attack. Since normal samples have lower LCR under mutated models in general, they would need more mutations than adversarial samples to draw a conclusion.

V. Related works

This work is related to studies on adversarial sample generation, detection and prevention. There are several lines of related work in addition to those discussed above.

a) Adversarial training: The key idea of adversarial training is to augment training data with adversarial examples to improve the robustness of the trained DNN itself. Many attack strategies have been invented recently to effectively generate adversarial samples like DeepFool [30], FGSM [13], C&W [6], JSMA [34], black-box attacks [32] and others [40], [37], [10], [5], [51]. However, adversarial training in general may overfit to the specific kinds of attacks which generate the adversarial samples for training [28] and thus can not guarantee robustness on new kinds of attacks.

b) Adversarial sample detection: Another direction is to automatically detect those adversarial samples that a DNN will mis-classify. One way is to train a 'detector' subnetwork from normal samples and adversarial samples [29]. Alternative detection algorithms are often based on the difference between how an adversarial sample and a normal sample would behave in the softmax output [55], [16], [24], [9] or under random perturbations [47].

c) Robustness metrics: Different metrics has been proposed in the machine learning community to measure and provide evidence on the robustness of a target DNN [54], [49]. Besides, in [35] and the following work [41], [25], neuron coverage and its extensions are argued to be the key indicators of the DNN robustness. In [3], they proposed adversarial frequency and adversarial severity as the robustness metrics and encode robustness as a linear program.

d) Testing and formal verification: Testing strategies including white-box [35], [45], black-box [50] and mutation testing [26] have been proposed to generate adversarial samples more efficiently for adversarial training. However, testing can not provide any safety guarantee in general. There are also attempts to formally verify certain safety properties against the DNN to provide safety guarantees [17], [19], [20], [48].

VI. Conclusion

In this work, we propose an approach to detect adversarial samples for Deep Neural Networks at runtime. Our approach is based on the evaluated hypothesis that most adversarial samples are much more sensitive to model mutation testing than normal samples in terms of label change rate. We then propose to detect whether an input sample is likely to be normal or adversarial by statistically checking the label change rate of an input sample under model mutation testing. We evaluated our approach on both MNIST and CIFAR10 dataset and showed that our algorithm is both accurate and efficient to detect adversarial samples.

REFERENCES

[1] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*, 2018.

[2] A.Wald. *Sequential Analysis*. Wiley, 1947.

[3] Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya Nori, and Antonio Criminisi. Measuring neural net robustness with constraints. In *Advances in neural information processing systems*, pages 2613–2621, 2016.

[4] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[5] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. *arXiv preprint arXiv:1712.04248*, 2017.

[6] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 39–57. IEEE, 2017.

[7] Nicholas Carlini and David Wagner. Audio adversarial examples: Targeted attacks on speech-to-text. *arXiv preprint arXiv:1801.01944*, 2018.

[8] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM, 2006.

[9] Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410*, 2017.

[10] Angus Galloway, Graham W Taylor, and Medhat Moussa. Attacking binarized neural networks. *arXiv preprint arXiv:1711.00449*, 2017.

[11] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai 2: Safety and robustness certification of neural networks with abstract interpretation. In *Security and Privacy (SP), 2018 IEEE Symposium on*, 2018.

[12] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[13] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[14] Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens van der Maaten. Countering adversarial images using input transformations. *arXiv preprint arXiv:1711.00117*, 2017.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[16] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136*, 2016.

[17] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification*, pages 3–29. Springer, 2017.

[18] Yue Jia and Mark Harman. An analysis and survey of the development of mutation testing. *IEEE transactions on software engineering*, 37(5):649–678, 2011.

[19] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

[20] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Towards proving the adversarial robustness of deep neural networks. *arXiv preprint arXiv:1709.02802*, 2017.

[21] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

[22] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[23] Axel Legay, Benoît Delahaye, and Saddek Bensalem. Statistical model checking: An overview. In *International conference on runtime verification*, pages 122–135. Springer, 2010.

[24] Shiyu Liang, Yixuan Li, and R Srikant. Enhancing the reliability of out-of-distribution image detection in neural networks.

[25] Lei Ma, Felix Juefei-Xu, Jiyuan Sun, Chunyang Chen, Ting Su, Fuyuan Zhang, Minhui Xue, Bo Li, Li Li, Yang Liu, et al. Deepgauge: Comprehensive and multi-granularity testing criteria for gauging the robustness of deep learning systems. *arXiv preprint arXiv:1803.07519*, 2018.

[26] Lei Ma, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Felix Juefei-Xu, Chao Xie, Li Li, Yang Liu, Jianjun Zhao, et al. Deepmutation: Mutation testing of deep learning systems. *arXiv preprint arXiv:1805.05206*, 2018.

[27] Xingjun Ma, Bo Li, Yisen Wang, Sarah M Erfani, Sudanthi Wijewickrema, Michael E Houle, Grant Schoenebeck, Dawn Song, and James Bailey. Characterizing adversarial subspaces using local intrinsic dimensionality. *arXiv preprint arXiv:1801.02613*, 2018.

[28] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[29] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations. *arXiv preprint arXiv:1702.04267*, 2017.

[30] Seyed Mohsen Moosavi Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, number EPFL-CONF-218057, 2016.

[31] Ian Goodfellow Reuben Feinman Fartash Faghri Alexander Matyasko Karen Hambardzumyan Yi-Lin Juang Alexey Kurakin Ryan Sheatsley Abhibhav Garg Yen-Chen Lin Nicolas Papernot, Nicholas Carlini. cleverhans v2.0.0: an adversarial machine learning library. *arXiv preprint arXiv:1610.00768*, 2017.

[32] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 506–519. ACM, 2017.

[33] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pages 372–387. IEEE, 2016.

[34] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pages 372–387. IEEE, 2016.

[35] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 1–18. ACM, 2017.

[36] Uri Shaham, Yutaro Yamada, and Sahand Negahban. Understanding adversarial training: Increasing local stability of neural nets through robust optimization. *arXiv preprint arXiv:1511.05432*, 2015.

[37] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1528–1540. ACM, 2016.

[38] Aman Sinha, Hongseok Namkoong, and John Duchi. Certifying some distributional robustness with principled adversarial training. 2018.

[39] Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. *arXiv preprint arXiv:1710.10766*, 2017.

[40] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. Attacking convolutional neural network using differential evolution. *arXiv preprint arXiv:1804.07062*, 2018.

[41] Youcheng Sun, Xiaowei Huang, and Daniel Kroening. Testing deep neural networks. *arXiv preprint arXiv:1803.04792*, 2018.

[42] Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. Concolic testing for deep neural networks. *arXiv preprint arXiv:1805.00089*, 2018.

[43] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[44] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *Computer Science*, 2013.

[45] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. *arXiv preprint arXiv:1708.08559*, 2017.

[46] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.

[47] Jingyi Wang, Jun Sun, Peixin Zhang, and Xinyu Wang. Detecting adversarial samples for deep neural networks through mutation testing. *arXiv preprint arXiv:1805.05010*, 2018.

[48] Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Duane Boning, Inderjit S Dhillon, and Luca Daniel. Towards fast computation of certified robustness for relu networks. *arXiv preprint arXiv:1804.09699*, 2018.

[49] Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, Dong Su, Yupeng Gao, Cho-Jui Hsieh, and Luca Daniel. Evaluating the robustness of neural networks: An extreme value theory approach. *arXiv preprint arXiv:1801.10578*, 2018.

[50] Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. Feature-guided black-box safety testing of deep neural networks. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 408–426. Springer, 2018.

[51] Chaowei Xiao, Jun-Yan Zhu, Bo Li, Warren He, Mingyan Liu, and Dawn Song. Spatially transformed adversarial examples. *arXiv preprint arXiv:1801.02612*, 2018.

[52] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. Mitigating adversarial effects through randomization. *arXiv preprint arXiv:1711.01991*, 2017.

[53] Wayne Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Mike Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. Achieving human parity in conversational speech recognition. *arXiv preprint arXiv:1610.05256*, 2016.

[54] Huan Xu and Shie Mannor. Robustness and generalization. *Machine learning*, 86(3):391–423, 2012.

[55] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*, 2017.

[56] Fuxun Yu, Zirui Xu, Yanzhi Wang, Chenchen Liu, and Xiang Chen. Towards robust training of neural networks by regularizing adversarial gradients. *arXiv preprint arXiv:1805.09370*, 2018.

[57] Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, and Yibo Xue. Droid-sec: deep learning in android malware detection. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 371–372. ACM, 2014.