Sensitive-Sample Fingerprinting of Deep Neural Networks

Zecheng He
Princeton University
zechengh@princeton.edu

Tianwei Zhang
No Affiliation
tianweiz@alumni.princeton.edu

Ruby Lee
Princeton University
rblee@princeton.edu

Abstract

Numerous cloud-based services are provided to help customers develop and deploy deep learning applications. When a customer deploys a deep learning model in the cloud and serves it to end-users, it is important to be able to verify that the deployed model has not been tampered with.

In this paper, we propose a novel and practical methodology to verify the integrity of remote deep learning models, with only black-box access to the target models. Specifically, we define `Sensitive-Sample` *fingerprints, which are a small set of human unnoticeable transformed inputs that make the model outputs sensitive to the model's parameters. Even small model changes can be clearly reflected in the model outputs. Experimental results on different types of model integrity attacks show that the proposed approach is both effective and efficient. It can detect model integrity breaches with high accuracy (>99.95%) and guaranteed zero false positives on all evaluated attacks. Meanwhile, it only requires up to* $103\times$ *fewer model inferences, compared to non-sensitive samples.*

1. Introduction

The past few years have witnessed the fast development of deep learning (DL). One popular class of deep learning models is Deep Neural Networks (DNN), which has been widely adopted in many artificial intelligence applications, such as image recognition [20, 25], natural language processing [11, 28], speech recognition [19, 13] and anomaly detection [29, 21].

To make it automatic and convenient to deploy deep learning applications, many IT corporations offer cloud-based services for deep learning model training and serving, usually dubbed as Machine Learning as a Service (MLaaS). For example, Google Machine Learning Engine [1], Microsoft Azure ML Studio [2] and Amazon SageMaker framework [3] enable customers to deploy their models online and release query APIs to end users. Customers are charged on a pay-per-query basis.

However, deploying deep learning tasks in MLaaS brings new security concerns. First, the model owner does not manage or have control over the actual model in the cloud any more. This gives adversaries opportunities to intentionally tamper with the remote models, to make it malfunction. Different attacks against model integrity have been proposed: e.g., DNN trojan attack [26, 17, 10], poisoning attack [7, 30, 34, 31], etc. These attacks have been shown to be practical in various DNN-based applications, e.g. autonomous driving [17, 26], user authentication [10] and speech recognition [26]. Figure 1 shows an example of attacking a deep learning based face recognition system: an adversary can insert a trojan into the authentication model by slightly modifying the face classifier. The compromised model can still give correct prediction results for original faces. However, it will mis-classify an arbitrary person with a specific pair of galsses as "A. J. Buckley". With this technique the adversary can easily bypass the authentication mechanism without being detected.

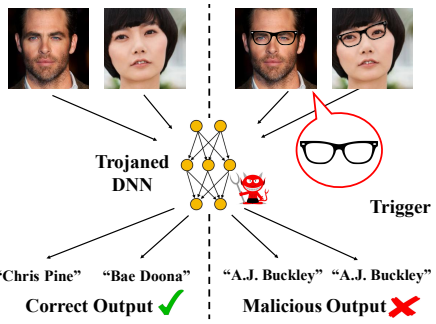


Figure 1: Illustration of a DNN trojan. A person without the trigger (left) is recognized correctly by the trojaned DNN. A person wearing a specific pair of glasses, i.e. the trigger, is mis-classified.

Second, a dishonest cloud provider may stealthily violate the Service Level Agreement (SLA), without making the customers aware, for financial benefits [35, 8]. For instance, the cloud provider can use a simpler or compressed model to replace the customers' models to save computational resources and storage [15]. Customers are annoyed with such SLA violations, even though it has a subtle impact

on the model accuracy, as they pay more for the resources than they actually get.

However, providing a methodology to protect the model integrity of DNN models deployed in clouds is challenging: (1) the complex cloud environment inevitably causes a big attack surface. (2) Once the customers submit their models to the clouds, the security status of the models are not transparent or directly verifiable to the customers. (3) For some model integrity attacks, the adversary only makes subtle modifications to the model, and wrong predictions only occur for specific attacker-chosen inputs which are imperceptible to the customers. (4) The cloud provider may not actively check the data integrity status in a timely manner. This gives adversaries opportunities to corrupt the models and cause damage before being detected.

In this paper, we are the first to show a new line of research where the integrity property of a DNN model can be dynamically verified by querying the model with a few carefully designed inputs. Specifically, we propose `Sensitive-Samples` fingerprinting, a new methodology for customers to verify the integrity of deep learning models stored in the cloud. The primary advantages of `Sensitive-Samples` are: ① high effectiveness and reliability, $> 99.95\%$ attack detection rate on all evaluated attacks, ② guaranteed zero false-positives, ③ high efficiency – although extensively querying the model with normal images may possibly detect the integrity breaches, it is very costly and inefficient on the pay-per-query basis. Our proposed approach achieves up to $103\times$ fewer model inferences and ④ requires only black-box accesses to the deployed model through APIs.

The key contributions of this paper are:

- We are the first using carefully designed transformed inputs as a defense, to protect the integrity property of DNNs.
- A novel and highly effective `Sensitive-Samples` generation approach for deep neural network integrity verification, achieving $> 99.95\%$ attack detection rate with only black-box accesses.
- A Maximum Active-Neuron Cover sample selection algorithm to generate the fingerprint of a DNN model from `Sensitive-Samples`, reducing the number of required model inferences by up to $103\times$.
- Comprehensive evaluation of our approach on different types of attacks on various applications and models.

The rest of the paper is organized as follows: Section 2 gives the background of deep neural networks, integrity attacks and defenses. Section 3 describes our new methodology of `Sensitive-Sample` fingerprinting. Section 4 introduces the experimental settings, datasets and attacks for evaluation. Section 5 gives the experimental results and discussions. We conclude the paper in Section 6.

2. Background and Related Work

2.1. Deep Neural Networks

A deep neural network (DNN) is a parameterized function $f_\theta : \mathcal{X} \mapsto \mathcal{Y}$ that maps an input $x \in \mathcal{X}$ to an output $y \in \mathcal{Y}$. A neural network usually consists of an input layer, an output layer and one or more hidden layers between the input and output. Each layer is a collection of units called *neurons*, connecting neurons in other layers.

The training process of a neural network is to find the optimal parameters θ that can accurately reflect the relationship between $\mathcal{X}$ and $\mathcal{Y}$. To achieve this, the user needs a training dataset $D^{train} = \{x_i^{train}, y_i^{train}\}_{i=1}^N$ with N samples, where $x_i^{train} \in \mathcal{X}$ is the input and $y_i^{train} \in \mathcal{Y}$ is the corresponding ground-truth label. Then a loss function L is adopted to measure the errors between the ground-truth output y_i^{train} and the predicted output $f_\theta(x_i^{train})$. The goal of training a neural network is to minimize this loss function (Eq (1)). After figuring out the optimal parameters θ^*, given a testing input x^{test}, the output $y^{test} = f_{\theta^*}(x^{test})$ can be predicted. This prediction is called `inference`.

$$\theta^* = \arg\min_\theta (\sum_{i=1}^{N} L(y_i^{train}, f_\theta(x_i^{train})) \tag{1}$$

2.2. DNN Integrity Attacks and Defenses

Neural network trojan attack. The attack goal is to inject a trojan into the model so that the model mis-classifies the samples containing a specific trigger [26, 17]. To achieve this, given a pretrained DNN model, the adversary carefully selects some "critical" neurons which the outputs are highly dependent on. He modifies the weights on the path from the selected neurons to the last layer by retraining the model using the data with triggers.

Targeted poisoning attack. The attack goal is to force the model to mis-classify a target class. The adversary achieves this by poisoning the dataset with carefully-crafted malicious samples. We consider two types of such attacks: the first one is error-generic poisoning attack [7, 30, 34], in which the outputs of the compromised model for the target class can be arbitrary. The second one is error-specific poisoning attack [31]: the adversary modifies the model to mis-classify the target class as a fixed class that he desires.

Model compression attack. The attacker's (cloud provider's) goal is to compress the DNN model with negligible accuracy drop, to save cloud storage for profit. There are different compression techniques to achieve this, e.g., pruning [18], quantization [16], low precision [12] and architecture optimization [24, 23].

Defenses. Past work have been designed to defeat model integrity attacks. For DNN trojan attacks, Liu et al. [27] proposed to detect anomalies in the dataset, or remove the trojan via model retraining or input preprocessing. For data

poisoning attacks, the typical solution is also to identify and remove the poisoning data from the dataset by statistical comparisons [9, 32]. While these methods are effective locally on white-box models, they fail to protect black-box models served in a remote MLaaS platform.

In the scenario of remote deep learning service, Ghodsi [15] proposed a protocol to verify if an untrusted service provider cheats the model owner with a simpler and less accurate model. However, this approach can only be applied to a specific class of neural networks with polynomial activation functions, and does not support max pooling.

3. Sensitive-Sample Fingerprinting

3.1. Overview

We consider the attack scenario in which the customer uploads a machine learning model f_θ to the cloud provider for model serving. However, an adversary may compromise the model and stealthily change it to $f_{\theta'}$. The customer wants to verify if the black-box model served by the cloud provider is actually the one he uploaded. Although extensively querying the model with normal images may detect the integrity breaches, it is very costly and inefficient on the pay-per-query basis.

Our main idea is that, we can carefully generate a small set of transformed inputs $\{v_i\}_{i=1}^n$, whose outputs predicted by `any` compromised model will be different from the outputs predicted by the original intact model. We call such transformed inputs `Sensitive-Samples`. We use a small set of these transformed inputs and their corresponding correct model outputs as the ***fingerprint*** of the DNN model, i.e. $\mathbb{FG} = \{(v_i, f_\theta(v_i))\}_{i=1}^n$.

To verify the integrity of a model, the customer first uses the correct model locally to generate `Sensitive-Samples` and obtain the corresponding output $y = f_\theta(v)$. For verification, he simply sends these samples to the cloud provider and obtains the output $y' = f_{\theta'}(v)$. By comparing y and y', the customer can check if the model is intact or changed.

There are some requirements in designing a good fingerprint, especially a good input transform, for integrity checking. We define a qualified fingerprint as one satisfying the following characteristics:

- **Effectiveness.** The fingerprint must be sensitive to even subtle modification of model parameters. In some attacks, the adversary changes a small number of parameters, e.g. selective neuron modification [26].
- **Efficiency.** The fingerprint must be light-weight and efficient, in order to reduce the cost and overhead for the verification, and avoid raising any suspicions.
- **Black-box verification.** The model served by the cloud provider is a black-box to the customer, thus the verification process must be feasible under this setting.
- **Hard to spot.** The generated fingerprint should look similar to natural inputs so the adversary cannot recognize if it is used for integrity checking, or for normal model serving.
- **Generalizable.** The fingerprint generation algorithm should be independent of the machine learning models, the training datasets and the attacks. It must be able to detect `any` unknown attacks.

3.2. Single Sensitive-Sample Generation

A DNN model can be defined as a function $y = f_\theta(x)$. Here θ is the set of all parameters in the model. We rewrite the model function as $y = f(W, x) = [y_1, ..., y_r]^T = [f_1(W, x), ..., f_r(W, x)]^T$. Here $W = [w_1, w_2, ..., w_s]$ is a subset of parameters-of-interest in θ in our consideration, containing the weights and biases.

We assume W in the correct model is modified by Δw, i.e. $W' = W + \Delta w$. The corresponding outputs of the correct and compromised model become $y = f(W, x)$ and $y' = f(W + \Delta w, x)$, respectively. In order to precisely detect this change through y and y', the "sensitive" input v should maximize the difference between y and y'.

$$\begin{aligned} v &= argmax_x \ ||f(W + \Delta w, x) - f(W, x)||_2 \\ &= argmax_x \ ||f(W + \Delta w, x) - f(W, x)||_2^2 \\ &= argmax_x \ \Sigma_{i=1}^r ||f_i(W + \Delta w, x) - f_i(W, x)||_2^2 \end{aligned} \tag{2}$$

where $||\cdot||_2$ denotes the l_2 norm of a vector. With Taylor Expansion:

$$f_i(W + \Delta w, x) = f_i(W, x) + \frac{\partial f_i(W, x)}{\partial W}^T \Delta w + O(||\Delta w||_2^2) \tag{3}$$

Note that we assume no prior-knowledge on Δw (how the adversary modifies the model). Consider Δw as a perturbation of W, we approximate Eq (3) to the first-order term:

$$||f_i(W + \Delta w, x) - f_i(W, x)||_2^2 \approx ||\frac{\partial f_i(W, x)}{\partial W}^T \Delta w||_2^2 \tag{4}$$

$$\propto ||\frac{\partial f_i(W, X)}{\partial W}||_2^2 \tag{5}$$

Note that the left-hand side of Eq (4) models the difference of output y_i between a correct DNN and a compromised DNN with weights perturbation Δw.

In Eq (5) we conclude that the l_2 norm of the gradient $||\frac{\partial f_i(W,x)}{\partial W}||_2$ can model the element-wise "sensitivity" of the DNN output corresponding to the parameters. Therefore, the sensitivity S of $f(W, x)$ can be defined as:

$$S = \Sigma_{i=1}^r ||\frac{\partial f_i(W, x)}{\partial W}||_2^2 = \left|\left| \frac{\partial f(W, x)}{\partial W} \right|\right|_F^2 \tag{6}$$

where $||\cdot||_F$ is the Frobenius norm [4] of a matrix. Eq (6) serves as the main objective function of our problem. In practice, there are auxiliary constraints on the sample.

Sample Correctness. In some cases, there are some requirements for the range of sample data, denoted as $[p, q]$.

For instance, all pixels must be in the range of [0, 255] for a valid image input.

Small Perturbation. In Section 3.1, we described a `Sensitive-Sample` should look like a normal input, to prevent the adversary from evading the integrity checking. So we add one more constraint: the generated sample is a small perturbation of a natural data v_0 sampled from the original data distribution $\mathcal{D}_X$, i.e. the difference of the generated sample and v_0 should not exceed a small threshold ϵ.

Eqs (7) summarize the objective and constraints of this optimization problem. The constraint set $[p,q]^m$ is a convex set, therefore we can use Projected Gradient Ascent [5] to generate v.

$$\begin{aligned} v = \underset{x}{\operatorname{argmax}} & \left\| \frac{\partial f(W,x)}{\partial W} \right\|_F^2 \\ \text{s.t. } & x \in [p,q]^m \\ & \|x - v_0\| \leq \epsilon \end{aligned} \tag{7}$$

We show a single `Sensitive-Sample` generation algorithm in Algorithm 1. Line 8 initializes the input with any sample from the natural data distribution $\mathcal{D}_X$. Line 10 sets up the element-wise loss function $||\frac{\partial f_i(W,x)}{\partial W}||_2^2$. Line 11 sets up the sample correctness constraints. Line 12 loops while v is still similar to the original initialization v_0. itr_max is set to avoid an infinite loop. Lines 14-17 apply a gradient ascent on the sensitivity, a.k.a. S in Eq (6). Line 18 projects v onto the sample correctness constraint set.

Algorithm 1 Generating a `Sensitive-Sample`

1: **Function** Sensitive-Sample-Gen(f, W, itr_max, ϵ, lr)
2: /* f: the target model */
3: /* W: parameters in consideration */
4: /* itr_max: maximum number of iterations */
5: /* ϵ: threshold for small perturbation constraints */
6: /* lr: learning rate in projected gradient ascent */
7:
8: v_0= Init_Sample()
9: $v, i = v_0, 0$
10: $l_k = \left\| \frac{\partial f_k(W,v)}{\partial W} \right\|_2^2, \quad k = 1, 2...N_{Output}$
11: Constraint_Set = $[p,q]^m$
12: **while** (($|v - v_0| \leq \epsilon$) && ($i < itr\_max$)) **do**
13: $\Delta = 0$
14: **for** ($k = 0; k < N_{Output}; k++$) **do**
15: $\Delta += \partial l_k / \partial v$
16: **end for**
17: $v = v + lr * \Delta$
18: v = Projection(v, Constraint_Set)
19: $i++$
20: **end while**
21: **return** $\{v, f(W,v)\}$

3.3. Fingerprint Generation: Maximum Active-Neuron Cover (MANC) Sample Selection

In some cases, a single `Sensitive-Sample` may not be enough to detect `any` weight changes. We observe that the main reason is that if a neuron is inactive[1] given an input sample, the sensitivity of all weights connected to that neuron becomes zeros, i.e. small modification of such weights will not be reflected in the outputs. We show the proof of this phenomenon in the extended version of this paper [22].

To address this problem, we propose Maximum Active Neuron Cover (MANC) sample selection algorithm to select a small number of samples from a bag of generated `Sensitive-Samples`, to avoid the inactive neurons. Our criterion is to minimize the number of neurons not being activated by any `Sensitive-Sample`, or equivalently, maximize the number of neurons being activated at least once by the selected samples. We call the resultant set of `Sensitive-Samples` with their corresponding model outputs, the ***fingerprint*** of the DNN model.

We can abstract it as a maximum coverage problem [6, 14]. As input, we are given a bag of generated `Sensitive-Samples` $B = \{S_1, ..., S_N\}$ and k, the number of desired samples. Suppose each `Sensitive-Sample` S_i activates a set of neurons P_i. The set $\{P_i\}$ may have elements (neurons) in common. We will select k of these sets such that a maximum number of elements (neurons) are covered, i.e. the union of the selected sets has maximal size.

We define the set of neurons being activated at least once by the k samples as *Active-Neuron Cover (ANC)*. It is the union of individually activated neurons P_i, i.e. $\bigcup_{i=1}^{k} P_k$. We would like to maximize the number of elements (neurons) in ANC, i.e. maximize $|\bigcup_{i=1}^{k} P_k|$.

Obtaining the accurate maximum of ANC is time-consuming and unnecessary in our experiment. Instead we use a greedy search to approximate the maximum. Intuitively, in each iteration t, we choose a set P_t which contains the largest number of uncovered neurons. We show the pseudo-code of MANC algorithm in Algorithm 2, and illustrate one step of the MANC algorithm in Figure 2. Line 5 in Algorithm 2 initializes the uncovered neurons to all neurons of interest, and the set of the selected samples to null. Line 9 computes the activations of neurons with corresponding input `Sensitive-Sample` $B[i]$. Line 10 determines the neurons that are activated by $B[i]$, i.e. P_i. Line 14 loops to select one sample in each iteration. Lines 16-21 determine which sample activates the largest number of uncovered neurons, and add it to the selected sample set. Line 22 updates the uncovered neurons.

[1] The neuron's output after the activation is 0 or very close to 0.

Algorithm 2 Maximum Active Neuron Cover (MANC) Sample Selection

1: **Function** MANC(Neurons, B, k)
2: /* Neurons: The neurons of interest */
3: /* B: The bag of samples from Algorithm 1 */
4: /* k: Number of desired samples */
5: Uncovered, Fingerprint = Neurons, []
6:
7: /* Each sample $B[i]$ activates neurons P_i */
8: **for** $(i = 0; i < |B|; i++)$ **do**
9: α = Activation(Neurons, B[i])
10: $P_i = \{\alpha_i | \alpha_i > 0\}$
11: **end for**
12:
13: /* Outer loop selects one sample each time */
14: **for** $(i = 0; i < k; i++)$ **do**
15: /* Inner loop among all samples to find the one that activates the largest number of uncovered neurons */
16: **for** $(j = 0; j < |B|; j++)$ **do**
17: NewCovered$_j$ = Uncovered $\bigcap P_j$
18: N_j = $| NewCovered_j |$
19: **end for**
20: l = $\text{argmax}_j \ N_j$
21: Fingerprint.add($B[l]$)
22: Uncovered = Uncovered - P_l
23: **end for**
24: **return** Fingerprint

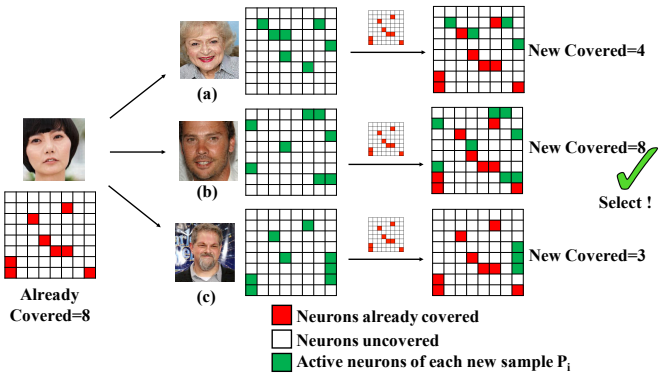


Figure 2: Illustration of selecting one sample in Algorithm 2 (line 16-21). Suppose the set $Fingerprint$ initially contains one selected sample (young lady, left). We want to select the next sample from three candidates (a),(b) and (c). We compute the neurons (red) that have been activated by the samples already in S, i.e. Active-Neuron Cover, and the uncovered neurons (white). We also compute the neurons activated by each candidate (P_i). Candidate samples (a),(b) and (c) activate 4,8 and 3 uncovered neurons, respectively. Thus we add the candidate (b) to $Fingerprint$ and update the covered neurons.

3.4. Model Output Specification

The form of the model output significantly affects the information that can be retrieved through black-box access. We consider three forms of y as the outputs of a DNN for classification tasks:

- Case 1: Numerical probabilities of each class.
- Case 2: Top-k (k>1) classification labels.
- Case 3: Top-1 classification label.

In general, the less information included in the output (from Case 1 (most) to Case 3 (least)), the harder it is to generate valid `Sensitive-Samples` and fingerprints. However, in our experiments, our proposed algorithm can detect an integrity breach for all known real attacks even if only the top-1 label is provided (Case 3) with high accuracy (>99.95%, <10 samples). Our experiments also show that we need even fewer samples (<3 samples) if more information is provided (Cases 1 and 2). We discuss these results in detail in Section 5.

3.5. Sensitive-Samples and Adversarial Examples

A similar and popular concept of our proposed `Sensitive-Samples` is adversarial examples [33]: the adversary intentionally adds human unnoticeable permutation Δx to a normal sample x, so the model gives a wrong prediction for this sample, i.e., $f_\theta(x + \Delta x) \neq f_\theta(x)$.

In this paper, we introduce `Sensitive-Samples`, another type of transformed inputs which also have human unnoticeable permutations from the normal samples, i.e., $z' = z + \Delta z$. Instead of making the model give wrong outputs, the outputs of the `Sensitive-Samples` change with the model parameters, i.e., $f_\theta(z') \neq f_{\theta+\Delta\theta}(z')$. Thus, unlike adversarial examples usually being used as an evasion attack strategy, `Sensitive-Samples` can be used as a powerful approach to defend against model integrity attacks. Table 1 shows the comparisons between our `Sensitive-Samples` and adversarial examples.

Table 1: Comparisons between `Sensitive-Samples` and adversarial examples.

	Sensitive-Samples	Adversarial-Examples
Similarity	Transformed inputs	
Purpose	Defense	Attack
Settings	Model parameters change $f_\theta(z') \neq f_{\theta+\Delta\theta}(z')$	Input perturbation $f_\theta(x + \Delta x) \neq f_\theta(x)$
Generation	White-box	White/Black box
Usage	Black-box	Black-box
Optimization Goal	Maximize the sensitivity of output w.r.t model parameters	Maximize the cost function*

* There are other approaches to generate adversarial examples.

4. Implementation

4.1. Attack Coverage

Our proposed method is generic and able to detect integrity breaches due to various attacks against DNN models. We evaluate this method on all four categories of real attacks in Section 2.2: neural network trojan attacks, error-generic and error-specific poisoning attacks and model compression attacks. These cover from subtle model changes to significant changes. We also consider the most general scenario: the adversary changes the weights of any arbitrary neurons to arbitrary values. The goal is to investigate the capability of our approach in defending against general model integrity breaches. We show the results of arbitrary weight changes in the extended version [22].

4.2. Datasets and Models

For most of the integrity attacks, we use the same datasets and models as in the literature. In Table 2, we list the model specifications, as well as the attack results.

Original accuracy denotes the accuracy of the original correct model. Attack goal shows the adversary's target of modifying the model. Note that we do not make any specific assumption about attack techniques, providing comprehensive protection against all types of model modification.

4.3. Hyper-parameters and Configurations

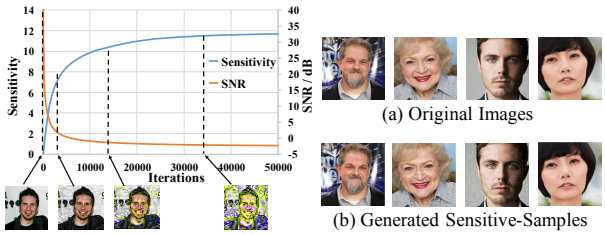
In our experiments, we set the learning rate to $1*10^{-3}$. We choose ADAM as our optimizer. We set itr_Max to 1000. We consider all the weights in the last layer as parameters-of-interest W. This is because the last layer must be modified in all existing attacks, and the output is most sensitive to this layer.

We reproduce the above four categories of DNN integrity attacks, and implement our solution using Tensorflow 1.4.1. We run our experiments on a server with 1 Nvidia 1080Ti GPU, 2 Intel Xeon E5-2667 CPUs, 32MB cache and 64GB memory. Under this setting, each `Sensitive-Sample` takes 3.2s to generate on average.

5. Evaluation

5.1. Sensitive-Sample Generation

We first show the generation mechanism and generated `Sensitive-Samples` in Figure 3 on VGG-Face dataset. Figure 3 left shows the trade-off between the sensitivity and similarity during the `Sensitive-Samples` generation process [2]. The blue line represents the sensitivity, i.e. defined in Eq (6) as $||\frac{\partial f(W,x)}{\partial W}||_F^2$. The orange line represents the similarity in terms of SNR. At the beginning of the optimization, the similarity is high, reflecting that the generated image is similar to the original input. However, the sensitivity is low, showing that the DNN output is not sensitive to the weight changes. It also indicates that directly using original images as fingerprints is not good. As the optimization goes on, the sensitivity increases significantly and finally converges to a high value. Meanwhile, artifacts are introduced in the sample generation, decreasing the similarity. In Figure 3 right, we show representative examples of the `Sensitive-Samples` on VGG-Face dataset.



(a) Original Images

(b) Generated Sensitive-Samples

Figure 3: Left: Sensitivity and similarity in the Sensitive-Sample generation process. Right: Original and generated Sensitive-Sample images for integrity checking on VGG Face dataset.

We show more generated `Sensitive-Samples` on CIFAR-10, GTSRB Traffic Sign and AT&T dataset in Figure 4, respectively. The generated images are very similar to the original inputs. Therefore, the attacker can hardly determine whether it is a natural image or a testing image for integrity checking. More generated $Sensitive-Samples$ can be found in the extended version [22].

5.2. Sensitive-Sample Effectiveness

We define a successful detection as "given N_S sensitive samples, there is at least one sample, whose top-1 label predicted by the compromised model is different from the top-1 label predicted by the correct model". Note that "top-1 label" is the most challenging case discussed in Section 3.4. In order to show the effectiveness of our approach more clearly, we show the missing rate (1-detection rate) of (1) Non-Sensitive Samples (green), (2) Sensitive-Samples + random selection (orange) and (3) Sensitive-Samples + MANC (blue) against four different attacks in Figure 5. In case (1), we randomly select N_S images from the original validation set. In case (2) and (3), we first generate a bag of 500 sensitive-samples and select N_S of them using random selection and MANC, respectively. We repeat the experiment 10,000 times and report the average missing rate.

We observe that Sensitive-Samples + MANC is highly effective in model integrity verification. In Table 3, for (a) neural network trojan attack, (b) error-generic poisoning attack and (c) error-specific poisoning attack, a fingerprint consisting of 3 Sensitive-Samples is enough to achieve a missing rate less than 10^{-4}. For (d) model

[2] The ϵ constraint in Eqs (7) is removed in Figure 3 left, to show the generation mechanism.

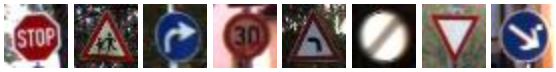
Table 2: Datasets and models in evaluation.

		Dataset	Task	Model	# Layers	# Conv layers	# FC layers	Original accuracy	Attack goal	Attack technique	Attack success rate
Neural network trojan attack		VGG-Face	Face recognition	VGG-16	16	13	3	74.8%	Misclassify inputs with triggers	Selective neural retraining	100%
Targeted poisoning	**Error-generic**	GTSRB	Traffic sign recognition	CNN	7	6	1	95.6%	Misclassify "Stop" traffic sign	Data poisoning	98.6%
	Error-specific	GTSRB	Traffic sign recognition	CNN	7	6	1	95.6%	Misclassify "Stop" to "Speed 100km"	Data poisoning	87.3%
Model compression		CIFAR-10	Image classification	CNN	7	6	1	87.59%	Save storage	Precision reduction	4x compression 86.94%
Arbitrary weights modification		AT&T	Face recognition	MLP	1	0	1	95.0%	General model modification	Arbitrary modification	$\star$

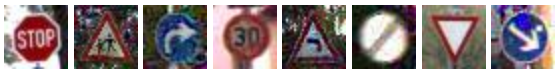
$\star$ We evaluate it for general integrity, thus no attack success rate.

(a) Original Images (CIFAR-10)

(b) Generated Sensitive-Samples (CIFAR-10)

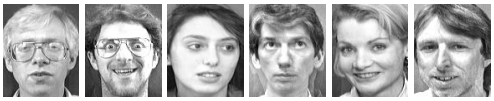


(c) Original Images (GTSRB Traffic Sign)



(d) Generated Sensitive-Samples (GTSRB Traffic Sign)

(e) Original Images (AT&T)



(f) Generated Sensitive-Samples (AT&T)

Figure 4: Original and generated Sensitive-Samples for integrity protection on CIFAR (a)(b), GTSRB Traffic Sign (c)(d) and AT&T (e)(f) dataset, respectively.

compression attack, although the compressed model is deliberately retrained to maintain accuracy on normal inputs, our `Sensitive-Sample` fingerprint still detects 99.96% integrity breaches (0.04% missing rate) with only 8 `Sensitive-Samples`. Further more, we compare the missing rate of Non-sensitive Samples, Sensitive-Samples + random selection and MANC in Figure 5. We observe that, `Sensitive-Samples` based approaches always achieve much lower missing rate than non-sensitive samples, regardless of N_S and attacks. Sensitive-Samples + MANC always achieves a lower missing rate than Sensitive-Samples + random selection, against all attacks.

Table 3: Missing rates (%) w.r.t to N_S on four real attacks.

Attacks $\setminus$ N_S	1	2	3	4	5	8
Neural Network Trojan Attack	5.93	0.22	0.00	0.00	0.00	0.00
Error-Generic Poisoning Attack	12.26	0.04	0.01	0.00	0.00	0.00
Error-Specific Poisoning Attack	2.20	0.01	0.00	0.00	0.00	0.00
Model Compression Attack	48.93	15.56	4.72	1.81	0.83	0.04

False Positives. Another advantage of our proposed solution is that false-positive is guaranteed to be zeros. Our proposed Sensitive-Samples defense leverages the determinacy of DNN model inference, therefore no false positive is raised. It is true for all the models and datasets we evaluate.

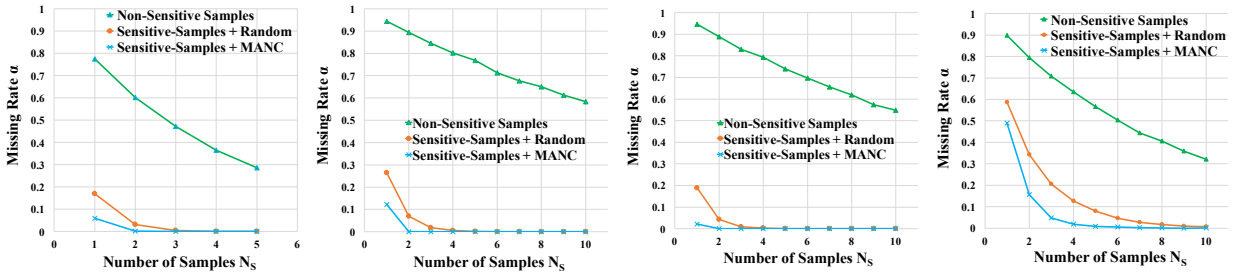
Output Specification. We evaluate the influence of the model output specification, e.g. top-k, numerical probabilities and digit precision. We list the missing rates corresponding to different output specifications (columns) and N_S (rows) in Table 4 against neural network trojan attacks. More results against other attacks are shown in the extended version [22]. "top-k" means the model outputs the k top labels. "p-dec-n" means the model outputs probabilities in addition to labels, with n digits after the decimal point. For example, "Top-1-p-dec-2" means the model outputs top-1 probability with the precision of 2 digits after the decimal point. Table 4 shows that, a large k, numerical probability and high precision of the probabilities embed more information in the output, and decrease the missing rate.

Table 4: Missing rates (%) w.r.t to the output specifications.

# of samples N_S	top-1	top-3	top-5	top-1-p-dec2	p-dec-1	p-dec-2
1	5.93	0.00	0.00	0.43	0.21	0.00
2	0.22	0.00	0.00	0.00	0.00	0.00
3	0.00	0.00	0.00	0.00	0.00	0.00

5.3. Sensitive-Sample Efficiency

In addition to the effectiveness in model integrity verification, our proposed approach is also highly efficient. We specifically consider minimizing the cost of verification, by reducing the number of required samples (model inferences). We show the required number of samples to achieve a given missing rate α against four real attacks in Table 5. We define the `Efficiency` as the ratio between the required number of samples (model inferences) between Non-Sensitive Samples and Sensitive-Samples + MANC. In or-



(a) (b) (c) (d)

Figure 5: Missing rate comparisons of different methods against (a) Neural Network Trojan Attack, (b) Error-Generic Poisoning Attack, (c) Error-Specific Poisoning Attack and (d) Model Compression Attack.

der to reveal subtle missing rates, we repeat our experiments in Section 5.2 10^8 times. Our proposed method significantly reduces the required number of samples, regardless of α, by up to $103\times$. Especially, our proposed approach is more comparatively efficient under small α, demonstrating it is of more obvious advantages in security-critical applications which require strict integrity verification.

Table 5: Required number of samples to achieve a given missing rate α against four real attacks. Our proposed method reduces required samples by up to 103x.

	Neural Network Trojan Attack						
Missing rate α	10^{-8}	10^{-7}	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}
Non-Sensitive Sample	74	65	56	47	38	28	21
Sensitive Sample	10	9	8	7	6	4	3
Sensitive Sample + MANC	4	4	3	3	3	2	2
Efficiency	**18.5x**	**16.5x**	**18.7x**	**15.6x**	**12.6x**	**14.0x**	**12.5x**
	Error-Generic Poisoning Attack						
Missing rate α	10^{-8}	10^{-7}	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}
Non-Sensitive Sample	332	291	249	208	166	125	83
Sensitive Sample	14	12	11	9	7	6	4
Sensitive Sample + MANC	4	4	4	4	3	2	2
Efficiency	**83.0x**	**72.8x**	**62.3x**	**52.0x**	**55.3x**	**62.5x**	**41.5x**
	Error-Specific Poisoning Attack						
Missing rate α	10^{-8}	10^{-7}	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}
Non-Sensitive Sample	309	270	232	193	155	116	77
Sensitive Sample	11	9	8	7	6	4	3
Sensitive Sample + MANC	3	3	3	3	3	2	2
Efficiency	**103.0x**	**90.0x**	**77.3x**	**64.3x**	**51.6x**	**58.0x**	**38.5x**
	Model Compression Attack						
Missing rate α	10^{-8}	10^{-7}	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}
Non-Sensitive Sample	502	439	376	314	252	189	126
Sensitive Sample	78	70	59	51	40	29	20
Sensitive Sample + MANC	31	31	30	28	25	18	8
Efficiency	**16.2x**	**14.2x**	**12.5x**	**11.2x**	**10.1x**	**10.5x**	**15.8x**

5.4. Resistance against Adversarial Fine-tuning

The adversary may attempt to evade our detection methodology. One possible strategy is that the adversary can generate the `Sensitive-Samples` from the intact model, and use these samples to fine-tune the compromised model. Then this fine-tuned model might make the customers' `Sensitive-Samples` used for verification insensitive. We call this potential evasive attack Adversarial Fine-tuning (AF).

We evaluate this evasive strategy with two model integrity attacks: error-generic poisoning and error-specific poisoning. Table 6 shows the detection missing rate using different numbers of verification `Sensitive-Samples` before and after fine-tuning. Note that because the customer can generate fingerprint from any arbitrary normal images, we assume the adversary fine-tunes the model with `Sensitive-Samples` different from the customer's.

It is interesting to note that the fine-tuning strategy cannot help the adversary evade the detection, and it actually makes the integrity checking easier. This is because `Sensitive-Samples` are designed to output very differently from the original model, thus fine-tuning on the `Sensitive-Samples` makes the tuned model deviate even more from the original model. This extra deviation can be more easily captured by other `Sensitive-Samples`.

Table 6: Missing rate (%) decreases as the attacker adversarial fine-tunes (AF) on Sensitive-Samples. It demonstrates that our proposed method is robust against more sophisticated attacks.

Attacks \ N_S	1	2	3	4	5
Error-generic poisoning (before AF)	12.26	0.04	0.01	0.00	0.00
Error-generic poisoning (after AF)	4.82	0.01	0.00	0.00	0.00
Missing rate increase	**-7.44**	**-0.03**	**-0.01**	**–**	**–**
Error-specific poisoning (before AF)	2.20	0.01	0.00	0.00	0.00
Error-specific poisoning (after AF)	0.02	0.00	0.00	0.00	0.00
Missing rate increase	**-2.18**	**-0.01**	**–**	**–**	**–**

6. Conclusion

In this paper, we show that the integrity of remote black-box deep learning model can be dynamically verified by querying the deployed model with a few carefully-designed human unnoticeable inputs and observing their outputs. Our proposed detection method defines and uses `Sensitive-Samples`, which introduce sensitivity of DNN outputs corresponding to the weights. Any small modification of the model parameters can be reflected in the outputs. Our evaluation on different categories of real DNN integrity attacks shows that our detection mechanism can effectively and efficiently detect DNN integrity breaches.

References

[1] https://cloud.google.com/ml-engine/docs/technical-overview, 2018.

[2] https://azure.microsoft.com/en-us/services/machine-learning-studio/, 2018.

[3] https://aws.amazon.com/sagemaker/, 2018.

[4] http://mathworld.wolfram.com/FrobeniusNorm.html, 2018.

[5] https://www.stats.ox.ac.uk/~lienart/blog-opti-pgd.html, 2018.

[6] A. A. Ageev and M. I. Sviridenko. Approximation algorithms for maximum coverage and max cut with given sizes of parts. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 17–30. Springer, 1999.

[7] B. Biggio, B. Nelson, and P. Laskov. Poisoning attacks against support vector machines. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, pages 1467–1474. Omnipress, 2012.

[8] K. D. Bowers, M. Van Dijk, A. Juels, A. Oprea, and R. L. Rivest. How to tell if your cloud files are vulnerable to drive crashes. In *ACM conference on Computer and communications security*, 2011.

[9] M. Charikar, J. Steinhardt, and G. Valiant. Learning from untrusted data. In *Annual ACM SIGACT Symposium on Theory of Computing*. ACM, 2017.

[10] X. Chen, C. Liu, B. Li, K. Lu, and D. Song. Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning. *ArXiv e-prints:1712.05526*, Dec. 2017.

[11] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[12] M. Courbariaux, Y. Bengio, and J.-P. David. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*, 2014.

[13] G. E. Dahl, D. Yu, L. Deng, and A. Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on audio, speech, and language processing*, 20(1):30–42, 2012.

[14] U. Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

[15] Z. Ghodsi, T. Gu, and S. Garg. Safetynets: Verifiable execution of deep neural networks on an untrusted cloud. In *Advances in Neural Information Processing Systems*, 2017.

[16] Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

[17] T. Gu, B. Dolan-Gavitt, and S. Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *CoRR*, abs/1708.06733, 2017.

[18] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations*, 2016.

[19] A. Y. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng. Deep Speech: Scaling Up End-to-end Speech Recognition. *CoRR*, abs/1412.5567, 2014.

[20] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. *CoRR*, abs/1512.03385, 2015.

[21] Z. He, A. Raghavan, S. Chai, and R. Lee. Detecting zero-day controller hijacking attacks on the power-grid with enhanced deep learning. *arXiv preprint arXiv:1806.06496*, 2018.

[22] Z. He, T. Zhang, and R. B. Lee. Verideep: Verifying integrity of deep neural networks through sensitive-sample fingerprinting. *arXiv preprint arXiv:1808.03277*, 2018.

[23] M. G. Hluchyj and M. J. Karol. Shuffle net: An application of generalized perfect shuffles to multihop lightwave networks. *Journal of Lightwave Technology*, 9(10):1386–1397, 1991.

[24] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

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

[26] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang. Trojanning attack on neural networks. In *25nd Annual Network and Distributed System Security Symposium, NDSS'18 , San Diego, California, USA, February, 2018*.

[27] Y. Liu, Y. Xie, and A. Srivastava. Neural trojans. In *IEEE International Conference on Computer Design*, 2017.

[28] M. Luong, H. Pham, and C. D. Manning. Effective Approaches to Attention-based Neural Machine Translation. *CoRR*, abs/1508.04025, 2015.

[29] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal. Long short term memory networks for anomaly detection in time series. In *Proceedings*. Presses universitaires de Louvain, 2015.

[30] S. Mei and X. Zhu. Using machine teaching to identify optimal training-set attacks on machine learners. In *AAAI*, pages 2871–2877, 2015.

[31] L. Muñoz-González, B. Biggio, A. Demontis, A. Paudice, V. Wongrassamee, E. C. Lupu, and F. Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 27–38. ACM, 2017.

[32] J. Steinhardt, P. W. W. Koh, and P. S. Liang. Certified defenses for data poisoning attacks. In *Advances in Neural Information Processing Systems*, 2017.

[33] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[34] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli. Is feature selection secure against training data poisoning? In *International Conference on Machine Learning*, pages 1689–1698, 2015.

[35] Y. Zhang, A. Juels, A. Oprea, and M. K. Reiter. Homealone: Co-residency detection in the cloud via side-channel analysis. In *2011 IEEE symposium on security and privacy*, pages 313–328. IEEE, 2011.