# NIC: Detecting Adversarial Samples with Neural Network Invariant Checking

Shiqing Ma, Yingqi Liu, Guanhong Tao, Wen-Chuan Lee, Xiangyu Zhang
Department of Computer Science, Purdue University
Email: {ma229, liu1751, taog, lee1938, xyzhang}@purdue.edu

***Abstract*—Deep Neural Networks (DNN) are vulnerable to adversarial samples that are generated by perturbing correctly classified inputs to cause DNN models to misbehave (e.g., misclassification). This can potentially lead to disastrous consequences especially in security-sensitive applications. Existing defense and detection techniques work well for specific attacks under various assumptions (e.g., the set of possible attacks are known beforehand). However, they are not sufficiently general to protect against a broader range of attacks. In this paper, we analyze the internals of DNN models under various attacks and identify two common exploitation channels: the provenance channel and the activation value distribution channel. We then propose a novel technique to extract DNN invariants and use them to perform runtime adversarial sample detection. Our experimental results of 11 different kinds of attacks on popular datasets including ImageNet and 13 models show that our technique can effectively detect all these attacks (over 90% accuracy) with limited false positives. We also compare it with three state-of-the-art techniques including the Local Intrinsic Dimensionality (LID) based method, denoiser based methods (i.e., MagNet and HGD), and the prediction inconsistency based approach (i.e., feature squeezing). Our experiments show promising results.**

## I. Introduction

Deep Neural Networks (DNNs) have achieved very noticeable success in many applications such as face recognition [41], self-driving cars [6], malware classification [11], and private network connection attribution [57]. However, researchers found that DNNs are vulnerable to adversarial samples [89]. Attackers can perturb a benign input (i.e., correctly classified input) so that the DNN would misclassify the perturbed input. Existing attack methods use two types of perturbation strategies: *gradient based approach* and *content based approach*. In the gradient based approaches, attackers view generating an adversarial sample as an optimization problem and conduct gradient guided search to find adversarial samples [10, 22, 42, 60, 68] (§II-C). In the content based approaches, attackers craft patches to inputs that are consistent with real world content of the inputs such as watermarks on images and black spots caused by dirt on camera lens to perturb the inputs [49, 72] (§II-C).

Real world applications, including many security critical applications, are trending to integrate DNNs as part of their systems. For example, iPhone X uses a face recognition system for authentication (unlocking phone, authenticating purchase, etc.). Many companies, e.g., Google and Uber, are developing self-driving cars that use DNNs to replace human drivers. However, the existence of adversarial examples is a fatal threat to the thriving of these applications because misbehaved DNNs would incur severe consequences such as identity theft, financial losses, and even endangering human lives. Thus detecting or defending against DNN adversarial samples is an important and urgent challenge.

There are existing works aiming to defend against or detect adversarial samples. Defense techniques try to harden DNNs using various methods such as *adversarial training* [22] and *gradient masking* [24]. The former tries to include adversarial samples in the training set so that the hardened models can recognize them. This technique is effective when the possible attacks are known beforehand. The latter aims to mask gradients so that attackers can hardly leverage them to construct adversarial samples. Recently, attackers have developed more advanced attacks against this type of defense. Some other works [16, 23, 55, 56, 99] do not aim to harden or change the models but rather detect adversarial samples during operation. For example, Ma et al. [53] proposed to use the Local Intrinsic Dimensionality (LID), a commonly used anomaly detection metric to detect adversarial samples. Xu et al. [99] proposed to examine the prediction inconsistency on an original image and its transformed version with carefully designed filters. MagNet [55] and HGD [45] propose to train encoders and decoders to remove the added noises of the adversarial samples. More detailed discussion of recent advances in defending and detecting adversarial samples is presented later (§II-D).

In this paper, we analyze the internals of individual DNN layers under various attacks and summarize that adversarial samples mainly exploit two attack channels: the *provenance channel* and the *activation value distribution channel*. The former means that the model is not stable so that small changes of the activation values of neurons in a layer may lead to substantial changes of the set of activated neurons in the next layer, which eventually leads to misclassification. The latter means that while the provenance changes slightly, the activation values of a layer may be substantially different from those in the presence of benign inputs. We then propose a method NIC (Neural-network Invariant Checking) that extracts two kinds of invariants, the *value invariants* ($VI$) to guard the value channel and the *provenance invariants* ($PI$) to guard the provenance channel. Due to the uncertain nature of DNNs, neural network invariants are essentially probability distributions denoted by models. Constructing DNN value invariants is to train a set of models for individual layers to describe the activation value distributions of the layers from the benign inputs. Constructing

Network and Distributed Systems Security (NDSS) Symposium 2019
24-27 February 2019, San Diego, CA, USA
ISBN 1-891562-55-X
https://dx.doi.org/10.14722/ndss.2019.23415
www.ndss-symposium.org

provenance invariants is to train a set of models, each describing the distribution of the causality between the activated neurons across layers (i.e., how a set of activated neurons in a layer lead to a set of activated neurons in the next layer). At runtime, given a test input which may be adversarial, we run it through all the invariant models which provide independent predictions about whether the input induces states that violate the invariant distributions. The final result is a joint decision based on all these predictions.

We develop a prototype and evaluate it on 11 different attacks (including both gradient based attacks and content based attacks), 13 models, and various datasets including MNIST [43], CIFAR-10 [40], ImageNet [78] and LFW [34]. The experimental results show that we can effectively detect all the attacks considered on all the datasets and models, with consistently over 90% detection accuracy (many having 100% accuracy) and an acceptable false positive rate (less than 4% for most cases and less than 15% for ImageNet, a very large dataset). We also compare our approach with state-of-the-arts, LID [53], MagNet [55]/HGD [45], and feature squeezing [99]. The results show that our proposed method can achieve consistently high detection accuracy with few false positives, while the other three detectors can achieve very good results on some attacks but cannot consistently detect all different types of attacks. We make the following contributions:

- We analyze the internals of DNNs under various attacks and identify the two main exploit channels: the provenance channel and the value channel.
- We propose a novel invariant based technique to detect adversarial samples, including novel methods of extracting value invariants and provenance invariants.
- To the best of our knowledge, we are the first one to explore detecting content based adversarial samples, which have significant different attack patterns.
- We also evaluate the robustness of our technique by designing a strong white-box adaptive attack, in which the attacker has full knowledge of our technique. Compared to existing techniques, NIC significantly increases the difficulty of constructing adversarial samples, with 1.4x larger mean $L_2$ distortion and 1200x longer construction time for MNIST.
- We implement a prototype [14]. The evaluation results show that our method can detect the 11 types of attacks considered with over 90% accuracy (many 100%) and limited false positives. Comparing with three state-of-the-art detectors, NIC shows consistently good results for all these attacks whereas other techniques perform well on a subset.

## II. Background and Related Work

### A. Neural Networks

A DNN can be represented as a function $F : X \rightarrow Y$, where $X$ denotes the input space and $Y$ the output space. It consists of several connected layers, and links between layers are weighted by a set of matrices, $w_F$. The training phase of a DNN is to identify the numerical values in $w_F$. Data engineers provide a large set of known input-output pairs $(x_i, y_i)$ and define a loss function (or cost function) $J(F(x), y^*)$ representing the differences between a predicted result $F(x)$ and the corresponding true label $y^*$. The training phase is hence to minimize the loss function by updating the parameters using the *backpropagation* technique [43]. The training phase is governed by *hyper-parameters* such as the learning rate. In our settings (detecting adversarial inputs), models are given and thus the hyper-parameters are fixed. In the testing phase, a trained model is provided with unseen inputs $X_t$, and for each input $x_t \in X_t$, the classification model assigns its label to be $C(x_t) = \text{argmax}_i F_i(x_t)$, where $F_i(x_t)$ represents the probability of $x_t$ being recognized as class $i$. The classification result is correct if the predicated result $C(x_t)$ is the same as the correct label $C^*(x_t)$.

In this paper, we focus on $m$-class DNN classification models. For such models, a model output is a vector with $m$ elements, and each of them represents the probability of the input being in a specific class. We use notations from previous papers [10, 70] to define a neural network:

$$y = F(x) = \text{softmax}(Z(x))$$

, where $x \in X$ and $y \in Y$. In such models, $Z(x)$ is known as the $logits$ and the softmax function normalizes the values such that for an output vector $y \in \mathbb{R}^m$, $y_i \in (0, 1)$ and $\sum y_i = 1$, where $y_i$ represents the probability of the input being class $i$. The final output will be the class with the highest probability.

### B. Adversarial Samples

DNNs are vulnerable to adversarial samples [89]. Intuitively, an adversarial sample is an input that is very similar to one of the correctly classified inputs (or benign inputs), but the machine learning model produces different prediction outputs for these two inputs. Existing works try to generate adversarial samples in two different kinds of approaches: *gradient based approach* and *content based approach*.

**Gradient based approach.** Formally, given a correctly classified input $x \in X$ with class $C(x)$ (notice that $C^*(x) = C(x)$), we call $x'$ an adversarial sample if

$$x' \in X \wedge C(x') \neq C(x) \wedge \Delta(x, x') \leq \epsilon$$

, where $\Delta(\cdot)$ denotes a distance function measuring the similarity of the two inputs and $\epsilon$ is the adversarial strength threshold which limits the permissible transformations. The equation means that $x'$ is a valid input ($x' \in X$) which is very similar to $x$ ($\Delta(x, x') \leq \epsilon$), but the model gives a different prediction output ($C(x') \neq C(x)$). Such adversarial samples are usually referred to as *untargeted adversarial samples*. Another type of attack is known as the *targeted attack*. In such attacks, the output of the adversarial sample $C(x')$ can be a particularly wanted class. For a given correctly classified input $x \in X$ ($C(x) = C^*(x)$) and a target class $t \neq C(x)$, a targeted adversarial sample $x'$ satisfies

$$x' \in X \wedge C(x) \neq C(x') \wedge C(x') = t \wedge \Delta(x, x') \leq \epsilon.$$

Thus, finding/generating adversarial samples can be viewed as optimization problems:

$$\textbf{min}\ \ \Delta(x, x')\ \ \textbf{s.t.}\ \ C(x') = t \wedge x' \in X\ \ \textbf{(Targeted)}$$

$$\textbf{min}\ \ \Delta(x, x')\ \ \textbf{s.t.}\ \ C(x') \neq C(x) \wedge x' \in X\ \ \textbf{(Untargeted)}.$$

Recent and popular adversarial attacks usually use the $L_p$-norm distance metric as the $\Delta(\cdot)$ function. Three $p$ values are commonly used in adversarial attacks, leading to $L_0$, $L_2$ and $L_\infty$ attacks. Previous works [10, 55, 99] have detailed comparison between them. In summary, (1) $L_0$ *distance limits the number of dimensions an adversarial sample can alter, but* *does not limit the degree of change for each dimension*. As a result, such adversarial samples tend to have fewer perturbation regions, but the change of each region is significant. (2) $L_\infty$ *distance limits the maximum change in any dimension, but does not limit the number of altered dimensions*. That is to say, the generated adversarial samples tend to have a large number of altered regions, but the change of each region is not substantial. (3) $L_2$ *distance (Euclidean distance) bounds the accumulated change to achieve a balance of the number of altered dimensions and the degree of change in each dimension*.

**Content based approach.** Another way of generating adversarial samples is to leverage the input content and provide semantically consistent/restrained perturbations on an original input to simulate real world scenarios. For example, images may have black spots because of dirt on camera lens and attackers can intentionally add multiple black spots to an image to simulate this. Under such assumptions, the attackers do not need to calculate the $\Delta(\cdot)$ distance value, and avoid optimizing the distance. As such, many such adversarial samples are human recognizable but have large $\Delta(\cdot)$ distances. There are a number of such recent attacks [7, 15, 49, 72].

## C. Existing Attacks

In this section, we discuss 11 existing representative attacks for DNN models, including both gradient based attacks and content based attacks. Note that while there are adversarial attacks for machine learning models in general [5], we focus on adversarial samples on DNN models in this paper. In Figure 1, we show sample images of various attacks using the MNIST dataset [43] to facilitate intuitive understanding. For gradient based attacks, we show the generated adversarial samples as well as the perturbations made (i.e., the colored regions). We use yellow to highlight small changes and red to highlight large changes. The lighter the color, the less the perturbation. For the content based attacks, the boxed regions represent the newly added elements to the original image.

**Fast Gradient Sign Method (FGSM)**: Goodfellow et al. proposed the *fast gradient sign method* to efficiently find adversarial samples [22]. Existing DNNs usually use piece-wise linear activation functions such as ReLU. These functions are easy to optimize, making it feasible to train a DNN. For such models, any change of the input is propagated to later hidden layers till the output layer. As such, errors on inputs can accumulate and lead to misclassification. The FGSM attack is based on this analysis and assumes the same attack strength at all dimensions and hence an $L_\infty$ attack. Also, it is an untargeted attack. Formally, an adversarial sample is generated by using the following equation:

$$x' = x - \epsilon \cdot \text{sign}(\nabla_x J(F(x)))$$

, where $\nabla$ represents the gradient and $J(\cdot)$ is the cost function used to train the model. Essentially, it changes all pixels simultaneously at a fix scale along the gradient direction. It can quickly generate adversarial samples as it only needs the gradient sign which can be calculated by backpropagation. An adversarial sample for the attack is shown in Figure 1(b). Observe that it makes small changes to many pixels.

**Basic Iterative Method (BIM)**: The *basic iterative method* [42] is an improved version of the FGSM attack. It generates adversarial samples with many iterations, and in each iteration it updates the generated sample with the following equation:

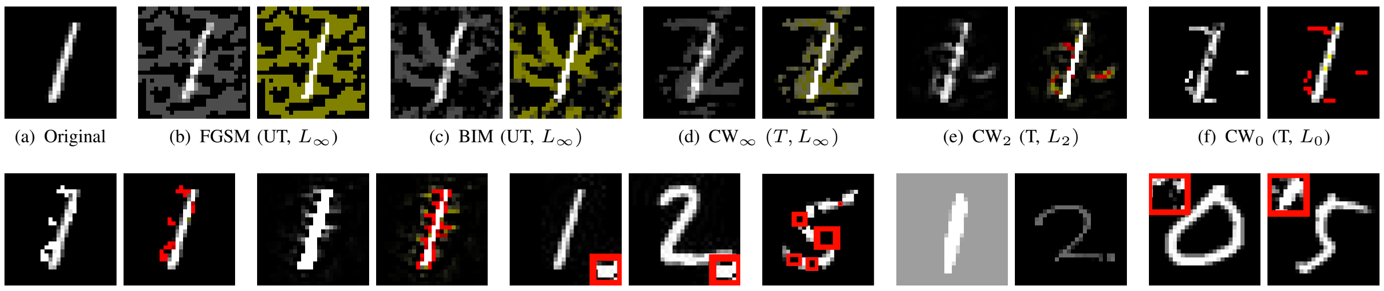
$$x'_i = x'_{i-1} - \text{clip}_\epsilon(\alpha \cdot \text{sign}(\nabla_x J(F(x'_{i-1}))))$$

, where $x'_0$ represents the original correctly classified input. In FGSM, a single step is taken along the direction of the gradient sign. BIM takes multiple steps, and in each iteration, it performs clipping to ensure the results are in the $L_\infty$ $\epsilon$-neighborhood of the original input. As such, it is also known as the Iterative FGSM (IFGSM). In practice, it can generate superior results than FGSM (Figure 1(c)).

**DeepFool**: Moosavi et al. designed the DeepFool attack [60] by starting from the assumption that models are fully linear. Under this assumption, there is a polyhedron that can separate individual classes. Composing adversarial samples becomes a simpler problem, because the boundaries of a class are linear planes and the whole region (for this class) is a polyhedron. The DeepFool attack searches for adversarial samples with minimal perturbations within a specific region by using the $L_2$ distance. The authors also adopt methods from geometry to guide the search. For cases where the model is not fully linear, the attack tries to derive an approximated polyhedron by leveraging an iterative linearization procedure, and it terminates the process when a true adversarial sample is found. In Figure 1(h), observe that the changes are in the vicinity of the original object. It is an untargeted attack.

**Jacobian-based Saliency Map Attack (JSMA)**: Papernot et al. proposed the *Jacobian-based Saliency Map Attack* [68] which optimizes $L_0$ distance using a greedy algorithm. The attack is an iterative process, and in each iteration, it picks the most important pixel to modify, and terminates when an adversarial sample is found. To measure the benefits of changing pixels, the attack introduces the concept of *saliency map*, which calculates how much each output class label will be affected when individual pixels are modified by using the Jacobian matrix. The attack then changes the pixel with the largest benefits (getting the target label). As an $L_0$ attack, JSMA modifies a very small number of pixels as shown in Figure 1(g), but the perturbation is more substantial than $L_\infty$ attacks such as FGSM or BIM. In the mean time, it has very high computation cost (due to the need of computing the Jacobian matrix), making it impractical for high dimension inputs, e.g., images from the ImageNet dataset.

**Carlini and Wagner Attack (CW)**: Carlini and Wagner proposed three different gradient based attacks using different $L_p$ norms, namely, $L_2, L_\infty$ and $L_0$. We refer to them as CW$_2$, CW$_\infty$ and CW$_0$ attacks, respectively. In the CW$_2$ attack, Carlini and Wagner are also solving the (targeted) optimization problem using the $L_2$-norm as the distance measurement. However, it features a few new designs to make it very effective. First of all, they use the logits $Z(\cdot)$ instead of the final prediction $F(\cdot)$ in the loss function. This design was shown to be critical for the robustness of the attack against defensive distillation methods [10]. Secondly, they introduced a variable $\alpha$ (also known as the optimal constant) to control the confidence of the adversarial samples so that they can get a better trade-off between the prediction and distance values. Another key design in this attack is that it maps the target variable to the *argtanh* space and then solves the optimization problem. This design enables the attack to use modern optimization solvers such as the Adam solver [39]. These techniques allow the CW$_2$ attack to very efficiently generate superior adversarial samples


(a) Original (b) FGSM (UT, $L_\infty$) (c) BIM (UT, $L_\infty$) (d) $CW_\infty$ ($T$, $L_\infty$) (e) $CW_2$ (T, $L_2$) (f) $CW_0$ (T, $L_0$)

(g) JSMA (T, $L_0$) (h) DeepFool (UT, $L_2$) (i) Trojan (T, CB) (j) Dirt(UT, CB) (k) Lighting (UT, CB) (l) Rectangle Patching (UT, CB)

Fig. 1: **Adversarial images of different attacks on MNIST.** *T: targeted. UT: untargeted. $L_0$, $L_2$ and $L_\infty$ stand for different distance metrics (gradient based attacks) and CB stands for content based attacks. Captions are the attack names. For gradient based attacks, Figure 1(a) is the original image. For each attack, the first image is adversarial and the second image highlights the perturbations. For content based attacks, red boxes highlight the added contents.*

as shown in Figure 1(e). Comparing with the other $L_2$ attack DeepFool, $CW_2$ perturbs fewer regions with smaller changes on most pixels. Most adaptive attacks are based on similar approaches (see §III-E and §IV).

**DeepXplore Attack**: In [72], Pei et al. observed that not all neurons are activated during testing, and these (inactivated) neurons can contain unexpected behaviors. Thus they proposed a systematic way of generating adversarial samples to activate all neurons in the network so that it can uncover unexpected behaviors in DNNs. The proposed attacks are leveraging semantic information (known as domain-specific constraints) to perform the perturbation. Specifically for the image type of inputs, they propose three methods: the *Lighting attack, Rectangle Patching attack* and the *Dirt attack*.

In the *Lighting attack*, the attackers change all the pixels' values by the same amount to make them darker or brighter. This action simulates the images taken under different lighting conditions (Figure 1(k)). In the *Rectangle Patching attack*, a single small rectangle patch is placed on the original image to simulate that the camera lens are occluded. Notice that in this attack, the content in the rectangle region are controlled by the adversary, and can be different for different images (Figure 1(l)). The *Dirt attack* adds multiple tiny black rectangles to the image for simulating the effects of dirt on camera lens (Figure 1(j)).

**Trojan Attack and Adversarial Patch Attack**: Liu et al. [49] proposed a systematic way to perform Trojan attacks on DNNs. The attacker first reverse-engineers the DNN to generate a well-crafted image patch (known as the *Trojan trigger*), and then retrains a limited number of neurons with benign inputs and trojaned inputs (i.e., benign inputs patched with the Trojan trigger, marked with the target output label) to construct the trojaned DNN. The trojaned DNN will perform the trojaned behavior (i.e., misclassifying images to the target label) for trojaned inputs, while retaining a comparable (sometimes even better) performance on benign inputs. This attack essentially simulates many real world scenarios such as images patched with watermarks or stamps.

Brown et al. [7] presented a method to create *adversarial patches* (conceptually similar to Trojan triggers) which achieves similar effects with the Trojan attack mentioned above. The difference is that it does not require retraining. From the perspective of detecting adversarial samples, there is no difference between this attack and the Trojan attack, as we are not aware if a model is trojaned and what the trigger is. Thus we consider them as the same type of attack.

## *D. Existing Defense and Detection*

**Existing Defense.** Defense techniques try to harden the NN models to prevent adversarial sample attacks [77, 86, 98]. Papernot et al. [69] comprehensively studied existing defense mechanisms, and grouped them into two broad categories: *adversarial training* and *gradient masking*. Goodfellow et al. introduced the idea of adversarial training [22]. It extends the training dataset to include adversarial samples with ground truth labels. There are also similar ideas that try to integrate existing adversarial sample generation methods into the training process so that the trained model can easily defend such attacks [54]. However, it requires prior knowledge of the possible attacks and hence may not be able to deal with new attacks.

The basic idea of gradient masking is to enhance the training process by training the model with small (e.g., close to 0) gradients so that the model will not be sensitive to small changes in the input [24]. However, experiments showed that it may cause accuracy degradation on benign inputs. Papernot et al. introduced *defensive distillation* to harden DNN models [70]. It works by smoothing the prediction results from an existing DNN to train the model and replacing the last softmax layer with a revised softmax-like function to hide gradient information from attackers. However, it was reported that such models can be broken by advanced attacks [8, 10, 67]. Athalye et al. [3] also showed that hardening or obfuscating gradients can be circumvented by gradient approximation. Papernot et al. [69] concluded that controlling gradient information in training has limited effects in defending adversarial attacks due to the transferability of adversarial samples, which means that adversarial samples generated from a model can be used to attack a different model.

**Existing Detection.** Adversarial sample detection is to *determine if a specific input is adversarial.* Many previous researches have studied to build detection systems [4, 21, 23, 29, 44, 56, 63, 65, 76]. We classify existing state-of-the-art works into three major categories.

*Metric based approaches.* Researchers have proposed to perform statistical measurement of the inputs (and activation values) to detect adversarial samples. Feinman et al. [16]

proposed to use the *Kernel Density estimation* (KD) and *Bayesian Uncertainty* (BU) to identify adversarial subspace to separate benign inputs and adversarial samples. Carlini and Wagner [9] showed this method can be bypassed but also commented this method to be one of the promising directions. Safetynet [51] quantized activations in late-stage ReLU neurons and used SVM-RBF to find the patterns that distinguish adversarial and natural samples. This approach is only tested on FGSM, BIM and DeepFool and requires adversarial samples to find the patterns. Inspired by ideas from the anomaly detection community, Ma et al. [53] recently proposed to use a measurement called *Local Intrinsic Dimensionality* (LID). For a given sample input, this method estimates a LID value which assesses its space-filling capability of the region surrounding the sample by calculating the distance distribution of the sample and a number of neighbors for individual layers. The authors empirically show that adversarial samples tend to have large LID values. Their results demonstrate that LID outperforms BU and KD in adversarial sample detection and currently represents the state-of-the-art for this kind of detectors. A key challenge of these techniques is how to define a high-quality statistical metric which can clearly tell the difference between clean samples and adversarial samples. Lu et al. [52] have shown that LID is sensitive to the confidence parameters deployed by an attack and vulnerable to transferred adversarial samples. Our evaluation in §IV also shows that although LID is capable of detecting many attacks, it does not perform well for a number of others.

*Denoisers.* Another way of detecting adversarial samples is to perform a pre-process (denoiser) step for each input [24, 55, 79]. These approaches train a model or denoiser (encoders and decoders) to filter the images so that it can highlight or emphasize the main component in the image. Doing so, it can remove the *noises* of an image including those added by attackers and thus correct the classification result. For example, MagNet [55] uses detectors and reformers (trained auto-encoders and auto-decoders) to detect adversarial samples. The method has been shown to work well on many attacks. But it is only tested on small datasets like MNIST and CIFAR-10. Liao et al. [45] argued that these pixel guided denoisers (e.g., MagNet) do not scale to large images such as those in the ImageNet dataset. Thus they proposed a *high-level representation guided denoiser* (HGD) for large images and achieved state-of-the-art results on ImageNet. A limitation of this kind of techniques is that denoisers are essentially trained neural networks. Training them remains a highly challenging problem (e.g., very time-consuming). Also, they are end-to-end differentiable, making them potentially vulnerable to white-box attacks [2, 99]. Moreover, the quality of denoisers depends on the training dataset and collecting a high-quality training set is also very demanding.

*Prediction inconsistency based approaches.* Many other works are based on prediction inconsistency [13, 25, 90]. Tao et al. [90] proposed to detect adversarial examples by measuring the inconsistency between original neural network and neural network enhanced with human perceptible attributes. However, this approach requires human defined attributes for detection. The state-of-the-art detection technique *Feature Squeezing* [99] achieves very high detection rates for various attacks. The authors pointed out that DNNs have unnecessarily large input feature space, which allows an adversary to produce adversarial samples. They hence proposed to use *squeezing* techniques (i.e., reducing the color depth of images and smoothing the images) to generate a few *squeezed* images based on the seed input. Feature squeezing essentially limits the degree of freedom available to an adversary. Then the DNN model takes all the squeezed images and the original seed image, and makes predictions individually. Adversarial samples are detected by measuring the distance between the prediction vectors of the original seed input and each of the squeezed images. If one of the distances exceeds a threshold, the seed input is considered malicious. However, according to [99] and our experiments in §IV, the technique does not perform well on FGSM, BIM and some content based attacks on CIFAR and ImageNet. This is because its performance highly depends on the quality of the designed squeezers, which remains an open research challenge.

Furthermore, most existing works focus on detecting gradient based attacks. It is unclear if they can detect content based attacks such as DeepXplore and Trojaning.

## III. Design

In this section, we first explain existing attacks are essentially exploiting two channels (§III-A). Then we discuss that guarding these channels can be achieved by invariant checking (§III-B). Finally, we introduce our design details (§III-C).

### A. Observations about DNN Attack Channels

We study the internals of individual layers of DNNs under the various kinds of attacks discussed in §II-C. We identify they mostly exploit two channels in the models: the *provenance channel* and the *activation value distribution channel*.

**Exploiting the Provenance Channel.** In our discussion, we say a neuron is activated if its activation function (e.g., ReLU) returns a non-zero value. A (hidden) layer of a DNN can be considered as a function that takes the activated neurons from the previous layer, performs matrix multiplication with the weight values in the layer, and then applies an activation function to determine what neurons are activated in the layer. *We consider the relation that the activated neurons in the preceding layer lead to the activated neurons in a given layer the provenance of the layer.*

Many attacks entail changing provenance. Intuitively, given an adversarial sample $x'$ that is similar to $x$ of class $A$, if the goal is to make the model to misclassify $x'$ to $B$, the provenance of $x'$ is often different from the typical provenance of $A$ or $B$. This is usually due to the internal instability of the model such that small changes lead to a different set of activated neurons.

Figure 2 and Figure 3 present normal operations of a DNN. For simplicity, we only show three layers, two hidden layers $L1$ and $L2$ and the output layer (that classifies $A$ and $B$). Figure 2 shows an input belonging to $A$ and Figure 3 to $B$. The colored nodes represent activated neurons, and white nodes represent the inactivated neurons. Darker colors denote larger values. Observe that some neurons are only activated by inputs belonging to a certain class [61, 89], such as nodes 1 and 6 for class A and nodes 4 and 8 for class B. There are also a lot of neurons that are activated by both kinds of inputs such as neurons 2, 3, and 7. The gray regions across the first two layers

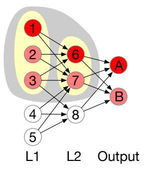


Fig. 2: A

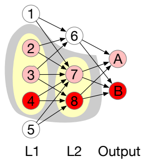


Fig. 3: B

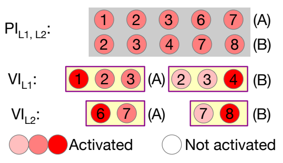


Fig. 4: $PI$ and $VI$

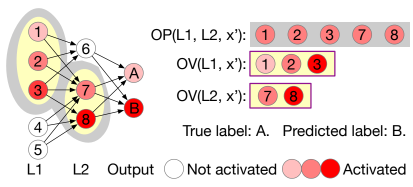


Fig. 5: Violating $PI$

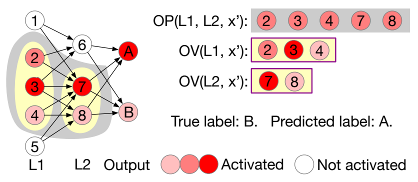


Fig. 6: Violating $VI$

denote the provenance. Observe that for $A$, neurons 1, 2, and 3 in $L1$ lead to 6 and 7 in $L2$; for $B$, neurons 2, 3, 4 lead to 7 and 8. Note that there are many possible provenance relations even for the same class. Our example is just for illustration.

Figure 5 shows the operation of an adversarial sample which is very similar to $A$ although it is misclassified as $B$. Observe neurons 1, 2, 3 in $L1$ lead to 7 and 8 in $L2$ and eventually the misclassified label $B$. Namely, the provenance is different from a typical $B$ provenance as in Figure 3. The root cause is that the model is very sensitive to small changes of neuron 1.

Trojan attack [49] is an example attack that exploits the provenance channel. The objective of the attack is that a trojaned model must not have performance degradation for benign inputs and the model should misclassify any input with a Trojan trigger to the target class. As such, it changes weight values at specific layers so that the provenance has substantial deviation in the presence of the Trojan trigger. A case study can be found in §IV-G. $L_0$ attacks also tend to exploit the provenance channel. These attacks introduce substantial changes to a limited number of elements in an input (see §II-C). As a result, different neurons (features) got triggered but they are not typical neurons (features) for the target class.

**Exploiting Activation Value Distribution.** Some attacks may not exploit the provenance channel. In other words, the provenance of an adversarial sample is no different from that of a benign input. In such cases, to cause misclassification, the activation value distribution of the activated neurons has to be different from that of a benign input. The exploitation can be illustrated by Figure 6, in which the adversarial input has the same provenance as a benign $B$ input (Figure 3). However, the substantially different activation values in $L2$ lead to the output value of class $A$ larger than $B$.

$L_\infty$ attacks (e.g., FGSM attack in Figure 1(b)) limit the change degree made to individual dimensions but do not limit the number of changed dimensions. For example, they tend to alter a lot of pixels, but the change of each pixel is small as shown in Figure 1. Thus they lead to substantially different value distributions in the first few layers. Note that in the first few layers, many neurons tend to be activated for both benign and adversarial inputs so that their provenance relations do not differ much.

Some adversarial samples exploit both channels such as $L_2$ attacks, which do not limit the number of perturbations or the degree of each perturbation, but rather constrain the total change using the Euclidean distance. Depending on the search procedure of an $L_2$ attack, it may exploit either value or provenance. Some attacks (e.g., DeepXplore) even exploit neurons that do not represent unique features for any class, leading to unique value distribution.

```
01: def fib(n):
02:    assert(n>=0)
03:    assert(from line 6 or 10)
04:    if n == 0 or n == 1:
05:      return n
06:    return fib(n-1)+fib(n-2)
07:
08: def main():
09:    x = input('Input a number:')
10:    print fib(x)
```

```
01: def dnn(x, M): # input:x, model:M
02:    L = M.layers
03:    for i in range(1, L.size):
04:      L[i] = L.w[i]*L[i-1]+L.b[i]
05:      for j in range(1, L[i].size):
06:        if L[i][j] < 0:
07:          L[i][j] = 0
08:      assert(PI, L[i], L[i-1])
09:      assert(VI, L[i])
10: return M
```

Fig. 7: Program invariant and DNN invariant

## B. Detection As Invariant Checking

The aforementioned DNN exploit channels are analogous to the ways that traditional attacks exploit software defects. Exploiting the provenance channel shares similarity to control flow hijacking [1], in which a regular execution path is hijacked and redirected to malicious payload. Exploiting the activation value distribution is analogous to state corruption [32], which represents a prominent kind of defective software behaviors that cause incorrect outputs but not necessarily crashes.

A classic approach to detecting software attacks is *invariant checking*. Specifically, to detect control flow hijacking, *control flow integrity* (CFI) techniques check control flow invariants (i.e., the control flow predecessor of an instruction must be one of a predetermined set of instructions). To detect state corruption, programmers explicitly add assertions to check pre- and post-conditions of individual program statements. Figure 7 shows an example of program invariants. It is a program that computes Fibonacci number. Observe that at line 2, the programmer uses an assert to ensure $n$ cannot be negative. Line 3 ensures that there are only two possible statements that can invoke the `fib()` function, namely, statements 6 and 10. Note that in many cases, such invariants are merely approximation due to various difficulties in program analysis (e.g., dealing with variable aliasing).

DNN computation is essentially a process of taking a model input and producing the corresponding classification output, which has a program structure like the one on the right of Figure 7. It iterates through the individual layers (lines 3 to 9). For each layer, it computes the activation values (line 4) from those of the previous layer, the weight of this layer (i.e., `L.w[i]`) and a bias factor `L.b[i]`. After that, an activation function (we use ReLU as an example, line 6-7 in Figure 7) is applied and a conditional statement is used to determine what neurons are activated in this layer (lines 5-7). Observe that exploiting the provenance channel is essentially altering the branch outcome of the conditional (line 6) while exploiting activation values is changing the distribution of the computed values at line 4.

Therefore, *our overarching idea is to check invariant violations during DNN computation* (lines 8 and 9 on the right). Our invariants are approximate as they cannot be precisely specified due to the uninterpretability of DNN. They are essentially probabilistic distributions constructed through learning. Different from other learning based techniques that

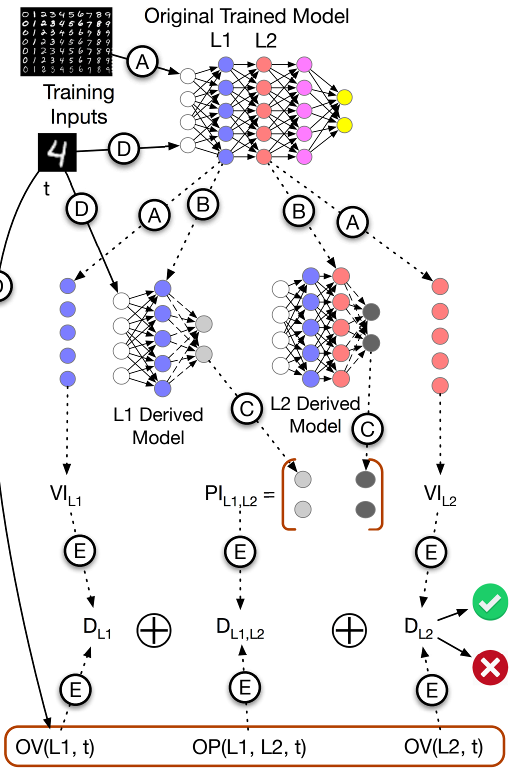


Fig. 8: Overview

require both benign and adversarial samples, *our invariants are only learned from benign samples and their computation*, and hence can provide general protection against a wide spectrum of attacks.

Intuitively, we define *the distribution of the activation values of two consecutive layers as the provenance invariant ($PI$) of the layers*. Later in §III-C, we show that learning such a distribution is too expensive so that we learn a reduced model. We further define *the activation value distribution of a given layer as its value invariant ($VI$)*. Figure 4 shows the $PI$s and $VI$s for the sample DNN (derived from benign inputs). Observe that the $PI$ of $L1$ and $L2$ is denoted as two vectors, representing neurons 1, 2, 3 leading to neurons 6, 7 (for class $A$), and neurons 2, 3, 4 leading to 7 and 8 (for class $B$). In Figure 5, the *observed provenance* ($OP$) for the adversarial input $x'$, which is misclassified as $B$, violates the provenance invariant (i.e., does not fit the distribution). In Figure 6, the *observed values* ($OV$) for layer $L1$ is substantially different from $VI_{L1}$ (denoted by their different colors), even though the $OP$ is consistent with $PI$. Note that our $VI$ technique is different from estimating LID [53]. The estimated LID measures the distances between a sample and a number of its neighbors whereas $VI$ trains models to explicitly describe activation value distributions. More importantly, as shown in the evaluation section, our $PI$ and $VI$ methods are complementary, together constituting an effective and robust solution outperforming LID.

## C. System Design

*1) Overview:* Figure 8 shows the overview of our approach. It is a training based approach, and we do not make any modification to the original trained model so that it does not affect the performance of the original model. We only use the benign inputs as our training data so that our technique is not specific to a certain attack. In step Ⓐ (Figure 8), we collect activation values in each layer for each training input. We then train a distribution for each layer for all benign inputs. These distributions (e.g., $VI_{L1}$ and $VI_{L2}$) denote the value invariants.

Ideally for $PI$s, we would train on any two consecutive layers to construct distributions of a concatenation of the activation values of the layers. However, a hidden layer may have many neurons, (e.g., 802,816 in one of our evaluated models). Such a distribution is often of high dimension (e.g., 2*802,816), very sparse and hence has limited predictive precision. Thus we train a reduced model that is of lower dimension and denser. Particularly, in Ⓑ, for each layer $l$ (e.g., $L1$, $L2$), we create a derived model by taking a sub-model from the input layer to $l$ and appending a new softmax layer with the same output labels as the original model. Observe that $L1$'s derived model has the input layer, $L1$ of the original model and a new softmax layer (dashed arrows). Then we freeze the sub-model weights, and re-train the softmax layer in the derived model. As it only trains one layer, it usually takes very limited amount of time to train a derived model. Intuitively, a derived model of layer $l$ is to predict the output class label based on the features extracted at $l$. Hence, a derived model of an earlier layer (e.g., $L1$) uses more primitive features while a derived model of a later layer (e.g., $L2$) uses more abstract features.

In step Ⓒ, we run each benign training input through all derived models, we collect the final outputs of these models (i.e., the output probability values for individual classes). For each pair of consecutive layers, we train a distribution for classification results of their derived models. The trained distribution is the $PI$ for these two layers. Essentially, we leverage the softmax functions in derived models to reduce the dimensions of the $PI$ models. Intuitively, we are computing an approximate notion of the provenance, namely, *how the prediction results may change from using features in a specific layer to using features in the next layer*.

For example, assume in a model that predicts a bird, bat, or a dog, an early layer extracts low level features such as beak, wing, tooth, feather, fur, tail, and claws whereas its next layer extracts more abstract features such as four-leg and two-wing. The derived model of the early layer would have a softmax that introduces strong connections from beak, feather, claws to bird; from wing, tooth, fur to bat; and from tooth, fur, claws, tail to dog. The derived model of the next layer associates four-leg to dog and two-wing to both bird and bat. The $PI$ model of the two layers hence would associate the prediction results of the two respective layers such as $\langle bat, bat\rangle$ (meaning the derived models of both layers predict bat), $\langle bat, bird\rangle$, and $\langle dog, dog\rangle$. Note that while the early layer may classify an input as bat and then the next layer classifies it as bird, the $PI$ model tells us that it is unlikely the early layer says dog whereas the later layer says bird. Essentially, the $PI$ model provides a way of summarizing the correlations between the features across the two layers. Ideally, we would like to train the $PI$ models to associate the primitive features to the corresponding abstract features (e.g., beak, feather to two-wing). However, such models would have input space of very high dimensions, much higher than our current design. Our results in §IV indicate our design is very effective.

At runtime (step Ⓓ), for each test input $t$ (e.g., the image of 4 in Figure 8), besides running the input through the original model, we also run it through all the derived models. We collect the activation values of each layer in the original model as the *observed values* ($OV$) (e.g., $OV(L_1, t)$ on the bottom of Figure 8) and the classification results of derived models of consecutive layers (in pairs) as the *observed provenance* ($OP$) (e.g., $OP(L_1, L_2, t)$). Then (step Ⓔ), we compute the probabilities $D$ that $OV$s and $OP$s fit the distributions of the corresponding $VI$s and $PI$s. All these $D$ values are aggregated to make a joint prediction if $t$ is adversarial.

In the remainder of the section, we discuss more details of individual components.

*2) Extracting DNN Invariants:* We extract two kinds of invariants: provenance invariants ($PI$) and activation value invariants ($VI$) from the internals of the DNN on the benign training inputs. These invariants are probabilistic, represented as distributions.

Recall that each layer of a DNN model is a function. We use $f_k$ to represent the $k$-th layer, and $f_k = \sigma(x_k \cdot w_k^T + b_k)$ where $\sigma$ is the activation function, $w_k$ is the model weight metrics, $b_k$ is the bias and $x_k$ is the input to this layer. Using these notations, a DNN with $n+1$ layers can be written as:

$$F = \text{softmax} \circ f_n \circ \ldots \circ f_1.$$

The output of layer $l$ is denoted as $f_l(x_l)$. The value invariant of layer $l$ is a function that describes the distribution of the activation values at $l$ for benign inputs. Hence, $0 \leq VI_l(x) \leq 1$ predicts if a model input $x$ is benign based on the activation values at layer $l$ . Hence, learning the weight $w$ of the $VI_l$ model is to solve the following optimization problem:

$$\textbf{min} \sum_{x \in X_B} J(f_l \circ f_{l-1} \circ \ldots \circ f_2 \circ f_1(x) \cdot w^T - 1)$$

, where $X_B$ represents the benign inputs and $J(\cdot)$ an error cost function. Intuitively, we aim to maximize the chance that $VI_l(x)$ predicts 1 for a benign input $x$.

As mentioned in §III-C, to compute $PI$s, we leverage derived models to reduce learning complexity and improve prediction accuracy. A derived model is constructed by taking the first few layers of the original model and then appending them with a new softmax layer. Thus the derived model $D_l$ for layer $l$ is the softmax layer, and $D_l$ can be defined as follows:

$$D_l = \text{softmax} \circ f_l \circ f_{l-1} \circ \ldots \circ f_2 \circ f_1.$$

$PI_{l,l+1}(x)$ predicts the probability of $x$ being benign based on the classification outputs of the derived models of $l$ and $l+1$ layers. Thus, training $PI_{l,l+1}$ (i.e., deriving its weights $w$) is an optimization problem as follows.

$$\textbf{min} \sum_{x \in X_B} J(\textbf{concat}(D_l(x), D_{l+1}(x)) \cdot w^T - 1)$$

, where **concat()** concatenates two vectors to a larger one.

*3) Training Invariant Models:* An important feature of our technique is that the invariant models (i.e., $VI$s and $PI$s) are trained from only benign inputs, which makes our technique a general detection approach different from existing training based approaches (see §II-D) that require adversarial samples in training and hence prior knowledge about the attack(s).

We model the training problem without adversarial samples as a *One-Class Classification* (OCC) problem. In OCC, most (or even all) training samples are positive (i.e., benign inputs in our context), while there will be all types of inputs (e.g., adversarial samples from various attacks in our context) during testing. OCC is a well studied problem [73, 91, 92]. Most existing classification algorithms can be extended for OCC. While OCC in general is not as accurate as techniques that leverage both positive and negative samples, it is sufficient and particularly suitable in our context as we use multiple invariant models to make joint decisions, which allows effectively mitigating inaccuracy in individual OCC models.

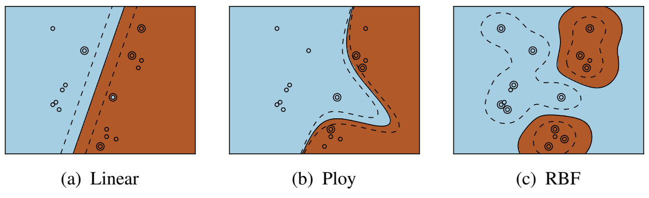

(a) Linear (b) Ploy (c) RBF

Fig. 9: Effects of different SVM kernels [71]

We use the *One-class Support Vector Machine* (OSVM) algorithm [92], which is the most popular classification algorithm for OCC problems and has a lot of applications [74]. The basic idea of OSVM is to assume a shape of the border between different classes (represented by different kernel functions), and calculate the parameters describing the border shape. Figure 9(a) uses a linear kernel function and the border of a class is assumed to be a line. Figure 9(b) uses a polynomial kernel and Figure 9(c) uses a radial basis function kernel (RBF). For OSVMs, the most widely used kernel is RBF. In our case, since most values in the input space are invalid (e.g., most random images are not realistic) and the valid inputs cluster in small sub-spaces, using the RBF kernel to achieve good accuracy. Moreover, although RBF is expensive, we train models for individual layers and consecutive layer pairs, which essentially breaks the entire invariant space into sub-spaces that have more regularity. While standard OSVM outputs a single number 0 or 1 to indicate if an input belongs to a group, we change the algorithm to output the probability of membership. In other words, we use the trained OSVM models to measure similarity.

During production runs, given an input $x$, we compute $VI_l(x)$ and $PI_{l,l+1}(x)$ for all possible $l$. In other words, we collect the probabilities of any invariant violations across all layers. A standard OSVM classifier is further used to determine if $t$ is an adversarial sample based on these probabilities and report the results. Details are elided.

## D. Randomization

Adversarial samples may be *transferable*, meaning that some adversarial samples generated for a model may cause misclassification on another model as well [66, 93]. Such a property makes it possible to attack a black-box system (assuming no knowledge about training data, method, process, hyper-parameters and trained weights). For example, Papernot et al. [66, 67] proposed to use synthesized data that are labeled by querying the target model to train a substitute model, and demonstrated that the generated adversarial samples for the substitute model can be used to attack the target model. Moosavi-Dezfooli et al. [59] found the existence of a universal perturbation across different images and demonstrated that such images can transfer across different models of ImageNet. Liu et al. [48] proposed an ensemble-based approach to generate transferable adversarial examples for large datasets such as ImageNet. Intuitively, a DNN model is an *approximation* (or fitting) of the real distribution. There is always a discrepancy between the approximation (the trained DNN model) and reality, which is the space of adversarial samples.

While our invariant checker essentially prunes the adversarial space, the fact that our checker is training based suggests that it is still an approximation. In other words, it is susceptible to transferable adversarial samples just like other detection techniques. Note that even proof-based techniques [19] suffer the same problem as they can only prove that inputs must satisfy certain conditions. They cannot prove a model is precise as having a precise model itself is intractable.

An effective and practical solution is to introduce randomization in the detectors. For example, MagNet [55] trains multiple encoders beforehand, and at runtime, they randomly select one encoder to use. In feature squeezing, Xu et al. also introduce random clip operations in their squeezer to make the attack harder. Even though theoretically such randomization mechanisms cannot prevent all possible attacks (especially when facing adaptive adversaries), they were shown to be very effective in practice. With randomization, attacks take much longer time and generate human unrecognizable inputs [55, 99].

As an integral part of our system, we also use randomization. Our randomization is based on the observation that DNNs make decisions by looking at all activated neurons rather than a subset of them [61, 89]. Due to the large number of contributing neurons, benign inputs are more robust to small neuron value changes than adversarial samples. Thus when we train $VI$s and $PI$s, we first apply a transformation function. It means that we use $g_l \circ f_l$ instead of $f_l$ as the input to train $VI$s and $PI$s. We prefer non-differentiable $g_l$ functions. Because if the attacker wants to craft adversarial samples under the assumption that the adversary knows the presence of detector, non-differentiable functions would significantly increase the difficulty [28] due to the lack of gradient information. In contrast, our technique directly uses the output of $g_l \circ f_l$ to train $VI$s and $PI$s, thus the introduction of $g_l$ does not affect our training. In this paper, typical $g_l$ functions include random scaling combined with discrete floor and ceiling functions. At runtime, we randomly select a set of $g_l$ functions and the corresponding trained invariant models in our detector. The results of randomization are shown in §IV.

## *E. Threat Model*

We assume that the adversary knows everything about the original classifier including the trained weights so that they can construct strong attacks such as the CW attacks, and the detector is not aware of the methods used for generating the adversarial samples. Depending on the information of the detector exposed to the adversary, there are multiple scenarios.

The weakest attack scenario is that the adversary knows nothing about the detector. In this case, the adversary generates the attack purely based on the original classifier. In §IV, we show that our detector can successfully detect a wide spectrum of such attacks.

The strongest adversary has full knowledge of our detector. Since the detector itself is also a classifier, this makes it vulnerable to adversarial samples too [20]. Note that this limitation is not specific to our technique as other existing detection techniques suffer the same problem too. Under such a strong threat model, our technique has better resilience compared to other techniques. In particular, as discussed in §III-D, we introduce randomization in our detector to improve its robustness. During training of the detectors, we first apply different transform functions $g_l$ on activated neurons to generate multiple randomized activation vectors. This allows us to have the flexibility of generating multiple detectors. At runtime, we can use different detectors (or their combinations) to detect adversarial samples. This substantially elevates the difficulty level of generating adversarial samples. In §IV, we show the effects of using the randomization technique. Note that complete prevention of adversarial samples is intractable for approximate models to which almost all practical DNNs belong. Our goal is to have a general and practical solution to substantially raise the bar for attackers.

## IV. Evaluation

In this section, we discuss the results of a large scale evaluation. Most experiments were conducted on two servers. One is equipped with two Xeon E5-2667 2.3GHz 8-core processors, 128 GB of RAM, 2 Tesla K40c GPU, 2 GeForce GTX TITAN X GPU and 4 TITAN Xp GPU cards. The other one has two Intel Xeon E5602 2.13GHz 4-core processors, 24 GB of RAM, and 2 NVIDIA Tesla C2075 GPU cards.

## *A. Setup*

**Datasets.** For gradient based attacks, we performed our experiments on three popular image datasets: MNIST [43], CIFAR-10 [40] and ImageNet [78]. MNIST is a grayscale image dataset used for handwritten digits recognition. CIFAR-10 and ImageNet are colored image datasets used for object recognition. For the ImageNet dataset, we used the ILSVRC2012 samples [35]. We chose these datasets because they were the most widely used datasets for this task and most existing attacks were carried out on them.

For content based attacks, Liu et al. evaluated Trojan attack on different datasets [49] (e.g., face recognition on LFW [34]). Besides using their face recognition data in our case study, we also ported the attack to two MNIST models, one Carlini model [10] and one Cleverhans model [64]. The trojaned Carlini model achieves 93% test accuracy on normal images and 100% attack accuracy, and the trojaned Cleverhans model has around 95% test accuracy on the original dataset with 100% attack accuracy. The evaluation of DeepXplore were conducted on their provided datasets (i.e., MNIST and ImageNet), their pre-trained models (i.e., LeNet-1, LeNet-4 and LeNet-5 for MNIST, and VGG16, VGG19 and ResNet50 for ImageNet), and their pre-generated adversarial samples on Github [72].

**Attack.** We evaluated our detection method on all the eleven attacks described in §II-C. For FGSM, BIM, and JSMA attacks, we used the implementations from the Cleverhans library [64] to generate adversarial samples, and for the other attacks, we used implementations from the authors [10, 49, 60, 72]. For the four targeted gradient based attacks (JSMA and three CW attacks), we evaluated on two different target settings: the next-class setting (denoted as *Next* in result tables) in which the targeted label is the next of the original label (e.g., misclassify an input of digit 2 to digit 3); and the least-likely class setting (denoted as *LL*), in which the target label is the most dissimilar to the original label (e.g., misclassify 1 to 8). Besides the four targeted attacks, the other seven included three untargeted gradient based attacks (FGSM, BIM and DeepFool) and four content based attacks (Trojaning, Dirt, Lighting and Rectangle Patching, with the last three from DeepXplore).

By default, we reused the same attack parameters used in feature squeezing [99] to construct attack images for gradient based attacks. DeepFool attack generates unrecognizable adversarial examples for the MNIST dataset and it is ignored by feature squeezing. As discussed in §II-C, the JSMA attack requires heavyweight computation and does not scale to large datasets like ImageNet. Thus we cannot test it on ImageNet. To generate Trojan triggers for the Trojan attack, we used the same configurations (e.g., size, shape and transparency) in their original paper [49]. DeepXplore targets to cover inactivate neurons, and it does not have extra attack parameters. For each dataset, we randomly partitioned the whole dataset to training data (60%, for training the detector), validation data (20%, for evaluating the detector with various $g$ functions mentioned in §III-D) and test data (20%, to test performance). For each attack, we generated 100 adversarial samples for each model.

**Models.** We evaluated our technique on thirteen popular models. These models are representatives of their kinds and used in the attacks under study [10, 49, 72]. For the MNIST dataset, we have collected six models from the work by Carlini et al. [10], the Cleverhans library (2 models) [64], and three different DNNs in the LeNet family [43], i.e., LeNet-1, LeNet-4 and LeNet-5. For the CIFAR-10 dataset, we used two models, one from Carlini and Wagner [10] and the other from DenseNet [12, 33]. For ImageNet, we used five models, ResNet50 [26], VGG16 [83], VGG19 [83], Inceptionv3 [88] and MobileNets [31, 58].

**Comparison.** We also compared NIC with other state-of-the-art detectors. For the metric based approach, we compared with LID [53]. Denoisers require a lot of parameter turning. Thus we chose to compare with well-tuned models, MagNet [55] for MNIST and CIFAR-10, and HGD [45] for ImageNet. For prediction inconsistency based techniques, we chose to compare with feature squeezing [99]. This paper only shows part of the comparison results due to the space limit. The remaining results are similar. For gradient based attacks on ImageNet, we used the Inceptionv3 model, and for MNIST and CIFAR-10, we adopted the same model used by Carlini and Wagner [10]. The Trojan attack on MNIST was also based on this model and the one from Cleverhans, and the attack on LFW was evaluated on the VGG19 model provided by the original authors. For DeepXplore attacks, the paper only shows the results for LeNet-4 and LeNet-5 on MNIST and VGG19 on ImageNet.

## B. Detecting Gradient Based Attacks

Table I shows the detection accuracy of our approach on gradient based attacks. Columns 1 and 2 show datasets and models. Column 3 shows the runtime overhead. It is measured by the average time used to run through the whole test dataset and adversarial samples (over 10 runs). Note that since we reuse the inner layer activation values of the original model when computing the results of $VI$s and $PI$s, invariant checking can be largely parallel with the original model execution. Thus the overhead is not significant. As shown in the table, for simple models (e.g., MNIST and CIFAR-10), we have less than 10% overhead. For complex models (e.g., ImageNet), it is less than 30%. Considering that the inference time for each image is very short (usually within a second), this is reasonable. Column 4 presents the false positive rates (FP) using default settings on each model, i.e., the number of benign inputs that are incorrectly detected as adversarial samples. The remaining columns show the detection results for each attack.

As shown in Table I, for the three MNIST models, NIC can detect all gradient based attacks with 100% accuracy with less than 3.8% false positive rate on benign inputs. Also the false positive rates of NIC on CIFAR-10 models are 3.8% and 4.2%, and the detection rates are all over 90%. For $L_\infty$ attacks on CIFAR-10, our approach can get 100% detection rates for all attack settings. For $L_2$ attacks, NIC can achieve nearly 100% detection rate on $CW_2$ attacks, and 91%/93% detection rates on DeepFool. The results of DeepFool are not 100% because it is an untargeted attack, which has much smaller perturbations compared to targeted attacks. For $L_0$ attacks, we can detect $CW_0$ and JSMA attacks with over 92% success rate.

For the ImageNet dataset, the detection rates show a similar pattern as the CIFAR-10 dataset. The images have a larger size and are more diverse, which makes the detection harder. For different models, the false positive rates range from 7.2% to 14.6%. In terms of detection rate, our detector can still achieve very high accuracy (over 90% for most cases).

## C. Detecting Content Based Attacks

We also evaluated our approach on content based approaches, namely the Trojan and DeepXplore attacks. The results are summarized in Table II. The above part shows the results of DeepXplore attacks, and the bottom part shows the results of the Trojan attack. The table has the same structure with Table I. The runtime overhead is similar and hence elided.

As shown in the table, NIC is able to detect all adversarial samples generated by the Trojan attack and DeepXplore attacks (100% for all cases) with a relatively low false positive rate (less than 4% on MNIST and LFW, less than 16% on ImageNet). More details of how they can be detected are described in §IV-D. Similar to the gradient based attacks, NIC has relatively high false positive rate on ImageNet than other datasets (e.g., MNIST), because of the large image size and content diversity. The results show that NIC is capable of detecting such content based attacks with high confidence and effectiveness.

## D. NIC *Invariants and Comparison*

In Table III, we show the comparison between NIC and other detectors (with gray background). Each row represents a detection method and the columns contain the results for different types of attacks (including both gradient based and content based attacks). For NIC, we also studied the effects of using only $PI$s or $VI$s on each attack to understand how they work. The rows are marked as $PI$ and $VI$ in Table III. From the table, we make the following observations.

**NIC Invariants.** For most $L_\infty$ attacks, using $VI$s alone has better results than using $PI$s alone, and using both produces the best results. This is because these attacks try to modify a

TABLE I: Performance on gradient based attacks

| Dataset | Model | OH | FP | FGSM | BIM | $CW_\infty$ |  | DF | $CW_2$ |  | $CW_0$ |  | JSMA |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  |  |  |  |  | Next | LL |  | Next | LL | Next | LL | Next | LL |
| MNIST | Cleverhans | 4% | 2.5% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% |
|  | Cleverhans2 | 3% | 3.8% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% |
|  | Carlini | 4% | 3.7% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% |
| CIFAR | Carlini | 9% | 4.2% | 100% | 100% | 100% | 100% | 91% | 96% | 100% | 98% | 100% | 94% | 100% |
|  | DenseNet | 8% | 3.8% | 100% | 100% | 100% | 100% | 93% | 97% | 100% | 97% | 100% | 92% | 99% |
| ImageNet | ResNet50 | 28% | 14.6% | 100% | 100% | 99% | 100% | 92% | 96% | 100% | 95% | 100% | - | - |
|  | VGG19 | 19% | 9.9% | 100% | 100% | 97% | 100% | 89% | 93% | 100% | 100% | 100% | - | - |
|  | Inceptionv3 | 21% | 13.8% | 100% | 100% | 99% | 100% | 90% | 96% | 100% | 100% | 100% | - | - |
|  | MobileNets | 18% | 7.2% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | - | - |

TABLE II: Content based attacks

| Dataset | FP | Model | Dirt | Lighting | RP |
|---|---|---|---|---|---|
| MNIST | 3.4% | LeNet-1 | 100% | 100% | 100% |
|  | 2.5% | LeNet-4 | 100% | 100% | 100% |
|  | 2.5% | LeNet-5 | 100% | 100% | 100% |
| ImageNet | 13.8% | ResNet50 | 100% | 100% | 100% |
|  | 15.2% | VGG19 | 100% | 100% | 100% |
|  | 15.9% | VGG16 | 100% | 100% | 100% |
| Dataset | FP | Model | Trojan | - | - |
| MNIST | 3.7% | Carlini | 100% | - | - |
|  | 2.5% | Cleverhans | 100% | - | - |
| LFW | 2.4% | VGG19 | 100% | - | - |

lot of pixels with a small degree of change. As a result, the $OV$s in the first few layers violate $VI$s significantly, which makes it easy to detect. On the other hand, as the features extracted by the first few layers are still very primitive, the generated $PI$s are not of high quality. For most $L_0$ attacks where the perturbations happen for a very limited number of pixels, but the per-pixel change is more substantial, using $PI$s alone produces good results. Using $VI$s are not as good because the number of activated neurons in each layer is large but the ones perturbed are in a small number. As such, the distance between $VI$s and $OV$s may not be sufficiently large. On the other hand, these perturbed neurons alter the set of activated neurons in later layers, leading to $PI$ violations. $L_2$ attacks bound the total Euclidean distance. As such, using $PI$s or $VI$s alone has mixed results, depending on which channel is being exploited. Our results indicate that either of them is (or both are) violated. Overall, using one kind of invariants alone is often not enough, and both kinds of invariants are very important to detecting adversarial samples.

For the Trojan attack, the adversarial samples tend to have the same activation pattern before the trojaned layer, and substantially change the prediction results at the trojaned layer, allowing them to be detected by $PI$s. Thus, Trojan attacks often lead to substantial invariant violations, which will be illustrated by a case study later in §IV-G. The DeepXplore attacks are designed to generate adversarial samples that try to activate inactivated neurons to uncover hidden behaviors. Thus, it often significantly violates the invariants. In most cases, such changes significantly alter the prediction results and violate the $PI$s. In the Lighting and Rectangle Patching attacks, the perturbations are substantial, leading to significant changes in the activation value distributions. Thus using $VI$s can detect most of them. In contrast, the Dirt attack adds a limited number of black dots to the image, and some changes are not significant. Thus using $VI$s alone cannot achieve very good performance.

**Denoisers.** Denoisers (i.e., MagNet and HGD) work well for $L_\infty$ attacks on all three datasets. In the mean time, we found that they do not perform well on $L_0$ attacks. This is because these denoisers are not guaranteed to be able to remove all the noise in an image, and the $L_0$ constraint limits the number of modified pixels and lowers the chance of being denoised. For most $L_2$ attacks, its performance is also not very good due to the same reason. As NIC also considers the effects of such noises (e.g., a few noisy pixels will change the activation pattern and violate the $PI$s), it can detect them as adversarial samples when the effects are amplified in hidden layers.

For content based attacks, this approach can detect all adversarial samples on MNIST and the RP attacks on ImageNet. But it does not perform well on other attacks. For Dirt attack which modifies relatively limited number of pixels, denoisers cannot remove some of the noises. For the Lighting attack, the whole color scheme is shifted with the same degree and denoisers do not perform well. In many cases they actually remove parts of the real objects. We believe it is because these parts have color patterns that are rare in the original training set, leading to some random behaviors of the denoisers. Denoisers show relatively good performance on the Trojan attacks as they transform the trigger to another representation, which lowers the chance to trigger the trojaned behaviors.

Denoisers tend to have relatively high false positive rates in most cases, meaning that the trained encoders and decoders affect the clean images especially for the colored datasets like ImageNet. Recall that denoisers are trained models. Training them is challenging and time-consuming. It may take more time than training the original classifier. One advantage of NIC is that it breaks the problem into small sub-problems. The detector consists of many small models and training them is much easier, making it more practical.

**LID.** For the gradient based attacks, the results of LID show that it is an effective approach for MNIST and CIFAR-10 with 93% and 90% average detection rates, respectively. However, it does not scale well on ImageNet, with 82% average detection rate. For the content based attacks, LID achieves relatively good results for both the Trojan attack and the DeepXplore attacks on the MNIST dataset, but relatively bad results on ImageNet. This is because images in ImageNet contain more noises and the clean images also have relatively large distances. This makes it more difficult identifying the boundaries between clean images and adversarial images. In the mean time, we observed that its false positive rates are relatively high for all cases (highest on ImageNet and LFW), meaning that it tends to mis-identify clean images as adversarial samples, which makes the detection results less confident.

**Feature Squeezing.** For gradient based attacks, feature squeezing achieves very good results for a set of attacks, but does not perform very well for FGSM, BIM and DeepFool attacks on colored datasets (CIFAR-10 and ImageNet). For extreme cases like FGSM attacks on CIFAR-10, the detection rate of feature squeezing is about 20%. According to [99], this is likely because that the designed squeezers are not suitable for such attacks on colored datasets. It illustrates a limitation of feature squeezing: requiring high quality squeezers for various models. To this end, our approach is more general.

Feature squeezing does not perform well on the Trojan attack with the detection rate ranging from 67% to 82%, which is

TABLE III: Comparison with state-of-the-art detectors

| Detector | Gradient-based | | | | | | | | | | | | | Content-based | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Dataset | FP | FGSM | BIM | $CW_\infty$ | | DF | $CW_2$ | | $CW_0$ | | JSMA | | Dataset | Model | FP | DeepXplore | | | Dataset | Model | FP | Trojan |
| | | | | | Next | LL | | Next | LL | Next | LL | Next | LL | | | | Dirt | Lighting | RP | | | | |
| $PI$ | MNIST | 1.8% | 84% | 92% | 94% | 99% | 95% | 99% | 96% | 100% | 100% | 100% | 100% | MNIST | LeNet-4 | 1.3% | 100% | 100% | 100% | MNIST | Carlini | 0.5% | 100% |
| $VI$ | | 1.9% | 100% | 100% | 100% | 100% | 93% | 100% | 95% | 90% | 88% | 85% | 83% | | | 1.2% | 100% | 100% | 100% | | | 3.2% | 76% |
| NIC | | 3.7% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | | | 2.5% | 100% | 100% | 100% | | | 3.7% | 100% |
| MagNet | | 5.1% | 100% | 100% | 96% | 97% | 91% | 85% | 87% | 85% | 85% | 84% | 84% | | | 4.3% | 100% | 100% | 100% | | | 5.6% | 100% |
| LID | | 4.4% | 97% | 96% | 93% | 92% | 92% | 92% | 91% | 92% | 92% | 96% | 96% | | | 4.2% | 100% | 100% | 100% | | | 4.5% | 100% |
| FS | | 4.0% | 100% | 98% | 100% | 100% | - | 100% | 100% | 91% | 94% | 100% | 100% | | | 3.9% | 97% | 39% | 72% | | | 4.0% | 82% |
| $PI$ | CIFAR | 2.6% | 52% | 64% | 68% | 63% | 69% | 89% | 89% | 98% | 100% | 94% | 100% | | LeNet-5 | 1.6% | 100% | 100% | 100% | | Cleverhans | 0.4% | 100% |
| $VI$ | | 1.2% | 100% | 100% | 100% | 100% | 78% | 82% | 88% | 63% | 68% | 58% | 62% | | | 0.9% | 100% | 100% | 100% | | | 2.1% | 72% |
| NIC | | 3.8% | 100% | 100% | 100% | 100% | 91% | 96% | 100% | 98% | 100% | 94% | 100% | | | 2.5% | 100% | 100% | 100% | | | 2.5% | 100% |
| MagNet | | 6.4% | 100% | 100% | 85% | 87% | 87% | 89% | 91% | 76% | 74% | 72% | 74% | | | 4.2% | 100% | 100% | 100% | | | 5.2% | 99% |
| LID | | 5.6% | 94% | 96% | 91% | 91% | 84% | 86% | 88% | 90% | 91% | 92% | 92% | | | 4.0% | 99% | 98% | 99% | | | 3.9% | 97% |
| FS | | 4.9% | 21% | 55% | 98% | 100% | 77% | 100% | 100% | 98% | 100% | 84% | 89% | | | 3.9% | 96% | 41% | 71% | | | 3.6% | 78% |
| $PI$ | ImageNet | 5.8% | 30% | 25% | 53% | 49% | 81% | 84% | 86% | 100% | 100% | - | - | ImageNet | VGG19 | 1.5% | 97% | 95% | 100% | LFW | VGG19 | 0.2% | 100% |
| $VI$ | | 1.4% | 99% | 100% | 99% | 100% | 53% | 90% | 92% | 29% | 12% | - | - | | | 1.0% | 43% | 100% | 93% | | | 2.2% | 46% |
| NIC | | 7.2% | 100% | 100% | 99% | 100% | 90% | 96% | 100% | 100% | 100% | - | - | | | 2.5% | 100% | 100% | 100% | | | 2.4% | 100% |
| HGD | | 9.7% | 97% | 95% | 92% | 92% | 83% | 83% | 85% | 81% | 82% | - | - | | | 6.2% | 88% | 69% | 100% | | | 4.7% | 80% |
| LID | | 14.5% | 82% | 78% | 85% | 86% | 83% | 78% | 80% | 79% | 80% | - | - | | | 9.6% | 83% | 89% | 91% | | | 4.8% | 75% |
| FS | | 8.3% | 43% | 64% | 98% | 100% | 79% | 92% | 100% | 98% | 100% | - | - | | | 3.9% | 89% | 40% | 82% | | | 3.3% | 67% |

much lower than using our method. Through manual inspection, it appears that after applying the squeezers, the effects of the Trojan triggers (which lead to misclassification) are degraded but not eliminated. As a result, many adversarial samples and their squeezed versions produce the same (malicious) classification results. For DeepXplore attacks, feature squeezing can detect almost all the adversarial samples generated by the Dirt attack, which adds black dots to the images to simulate the effects of dirt. This attack is very similar to $L_0$ attacks, on which feature squeezing performs very well. It cannot handle the Lighting attacks and Rectangle Patching (RP) attacks very well. The Rectangle Patching attack is very similar to the Trojan attack except that it generates a unique rectangle for each adversarial sample. Lighting attacks are similar to FGSM attacks but they change pixel values more aggressively. The changes can hardly be squeezed away.

We notice that the false positive rates of feature squeezing is relatively low. This shows that the designed squeezers are carefully chosen to avoid falsely recognize clean images as adversarial samples, which makes its results more trustworthy.

## *E. Adaptive Adversaries*

The experiments in the previous sections are to detect adversarial samples under the assumption that the attacker is not aware of the existence of the detector (black-box attack). As discussed in §III-D, we need to deal with a stronger threat model, in which the attacker knows everything of the dataset, the trained model and the trained detector including the method used to train them as well as the model parameters after training. Our method against this attack model is to train multiple detectors using different $g$ functions (§III-D). At runtime, NIC randomly chooses the detector(s) to use for the final decision. During this evaluation, we use three detectors and perform majority voting for the final decision.

We designed a $CW_2$ based white-box attack. The basic idea is to view the two classifiers (i.e., the original model and the detector) as a whole. In other words, we consider the output of the detector as part of the loss function to generate adversarial samples. The original $CW_2$ attack tries to minimize the $L_2$ distance of the generated adversarial sample and the original benign image such that the two images will lead to different prediction results. With the presence of NIC, the new attack modifies the optimization objective function. Now we minimize the $L_2$ distance of the two images as well as the $L_2$ distance of the activation neurons of the two images in each hidden layer and the prediction output vectors of the two images from all derived models (intuitively, minimizing $VI$ and $PI$ violations). We choose to make the attack untargeted, which tends to have less perturbation compared with targeted attacks. As the $g$ functions are not differentiable, we leverage a random process to find an adversarial sample that cannot be detected before the start of the optimization procedure.

If we limit the optimal constant (used to balance the amount of perturbation and the adversarial strength) to a rationale range, $[10^{-3}, 10^{6}]$, the same settings used in LID [53], the attack fails to generate adversarial images for all cases. This demonstrates that NIC is robust to such normal confidence attacks even under such strong white-box attack model. Without such limits, the attack can achieve 97% success rate on MNIST and CIFAR-10. The mean $L_2$ distortion of successful samples is 3.98 for MNIST and 2.76 for CIFAR-10, which is higher compared with other detectors (e.g., the $L_2$ distortion of feature squeezing is 2.80 on MNIST for adaptive, untargeted attacks [99]). This shows that our technique increases the difficulty of constructing adversarial samples.

Because of the complexity of the optimization objective function, it takes significantly longer time to generate adversarial samples. We performed an experiment on a desktop equipped with an i7-3770 3.40GHz CPU and 28 GB of RAM.

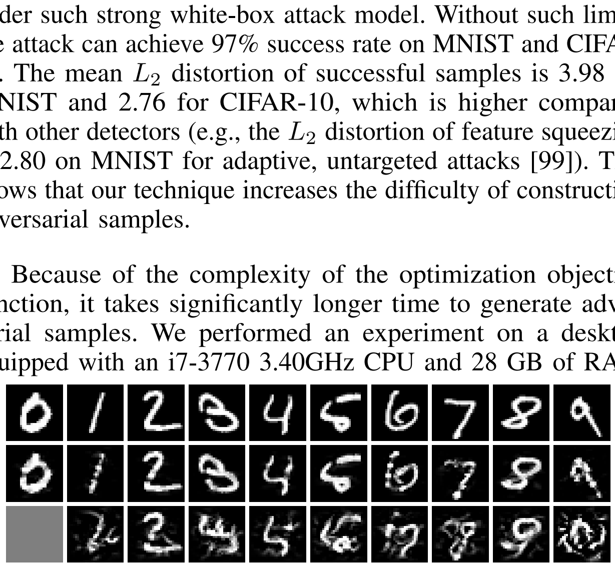


Fig. 10: **Adversarial Samples.** *1st line shows seed inputs, 2nd line shows without detector, 3rd line shows with 1 detector. Gray boxes mean no adversarial samples found in 1000 iterations.*

TABLE IV: Effects of different OCC algorithms

| Detector | FGSM | BIM | $CW_\infty$ Next | $CW_\infty$ LL | DF | $CW_2$ Next | $CW_2$ LL | $CW_0$ Next | $CW_0$ LL | JSMA | JSMA | Trojan | Dirt | Lighting | RP |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5NN | 14% | 11% | 9% | 10% | 4% | 5% | 4% | 3% | 1% | 1% | 2% | 26% | 12% | 2% | 24% |
| Linear | 22% | 20% | 17% | 19% | 8% | 12% | 15% | 8% | 12% | 22% | 24% | 32% | 34% | 93% | 45% |
| Poly | 89% | 86% | 79% | 82% | 85% | 88% | 89% | 84% | 85% | 84% | 86% | 94% | 89% | 99% | 95% |
| RBF | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% |

Specifically, generating a normal adversarial sample (without a detector) takes about 1 second, while generating an attack image takes about 420 seconds on average in the presence of one detector and around 1200 seconds on average in the presence of three detectors. This shows that such randomization approaches do increase the difficulty of generating adversarial samples. This result is also consistent with the findings in [28, 99]. Furthermore, the adversarial samples generated in the presence of a randomized detector usually have low confidence and are human unrecognizable. Figure 10 shows some generated attack images. The first row is the seed inputs, the second row shows the attack images without any detector and the last row shows the attack images to bypass one NIC detector. We can see that many of the last row images are not human recognizable.

### F. Choice of One Class Classifiers

In this part, we evaluated the effects of using different OCC algorithms. We compared four algorithms using the Carlini model on MNIST dataset: KNN-OCC (k=5), OSVM with linear kernel, OSVM with poly kernel and OSVM with the RBF kernel. The results are shown in Table IV. As we can see, KNN-OCC performs badly for this task, and the more complex the kernel is, the better results the OSVM algorithm can achieve. This is because complex kernels are more expressive and can identify the differences between clean images and adversarial samples. Potentially, the model accuracy may be further improved with better one-class classification algorithms.

### G. Case Study

We use the Trojan attack on a face recognition model (with 2622 output labels) to demonstrate invariant violations. The pre-trained VGG-face recognition model can be found at [49]. We use the same datasets (VGG and LFW) to generate adversarial samples to be consistent with the original paper [49]. Figure 11 shows the benign images (on the left of each subfigure) and the image patched with the Trojan trigger (on the right of each subfigure). The captions show the ground truth label. The trojaned model misclassifies all the images with the Trojan trigger to the target label `A.J. Buckley`. The attack triggers both value and provenance invariant violations at layer $L18$. The scale of the provenance invariant violation is more substantial. Table V shows part of the $PI$ for layers $L17$ and $L18$ (rows 3,4), and the three $OP$s for the three images with the Trojan trigger (i.e., the three rows on the bottom). Each row consists of the classification results of the derived model of $L17$ and the results of $L18$. Since it is difficult to visualize $PI$, we show a typical vector value in the $PI$ instead. Observe that the $OP$s are substantially different from the $PI$ with the differences highlighted in red. Intuitively, the $PI$ indicates that if $L17$ predicates `Mark` with a high confidence, $L18$ very likely predicts `Mark` with high confidence (slightly higher or lower than the previous layer result, no significant change) too. However, the first $OP$ says that $L17$ predicts `Mark` with a high confidence but $L18$ predicts `A.J. Buckley` with a high confidence. This is a clear violation. The three adversarial samples cause value invariant violations as well (Figure 12). The brown region describes the $VI$ distribution for $L18$. The adversarial samples (red dots) are clear outliers. However, NIC is not able to distinguish Trajoned models from those are simply vulnerable.

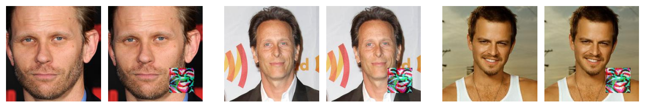

(a) Mark Pellegrino (b) Steven Weber (c) Carmine Giovinazzo

Fig. 11: Adversarial samples for the Trojan attack

| | A.J. Buckley | Mark Pellegrino | Steven Weber | Carmine Giovinazzo |
|---|---|---|---|---|
| | | $PI$ | | |
| L17 | 0.0001 | 0.8857 | 0.0001 | 0.0004 |
| L18 | 0.0001 | 0.9023 | 0.0001 | 0.0004 |
| | | $OP$ | | |
| L17 | 0.001 | 0.8734 | 0.0001 | 0.0004 |
| L18 | 0.9273 | 0.0001 | 0.0001 | 0.0001 |
| L17 | 0.0001 | 0.0001 | 0.7234 | 0.0001 |
| L18 | 0.9736 | 0.0001 | 0.0001 | 0.0001 |
| L17 | 0.0001 | 0.0001 | 0.0001 | 0.9243 |
| L18 | 0.9652 | 0.0001 | 0.0001 | 0.0004 |

TABLE V: $PI$ and $OP$s

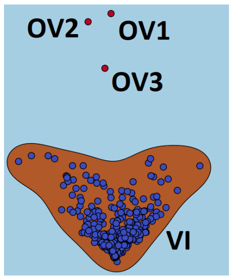


Fig. 12: $VI$ and $OV$s

## V. Related Work

We have covered many adversarial sample attacks, defense and detection works in §II. There are other types of attacks on machine learning models especially DNNs [27, 30, 82, 84, 85, 97]. Liu et al. [47] noticed that existing works study the perturbation problem by assuming an ideal software-level DNN, and argued this is not enough. They investigated adversarial samples by considering both perturbation and DNN model-reshaping (used by many DNN hardware implementations). Xiao et al. [96] utilized generative adversarial networks (GANs) to learn and approximate the distribution of original inputs so as to generate adversarial samples. Some works focus on applying poisoning attacks on machine learning models [36, 46, 80, 87]. Poisoning attacks aim at downgrading or compromising machine learning models by providing malicious training instances. Researchers have also proposed attacks to steal the parameters or hyper-parameters of machine learning models from open online services [94, 95]. Some works focus on studying the potential privacy leakage issues of machine learning [30, 37, 50, 62, 75, 84]. Fredrickson et al. [17, 18] proposed membership inference attacks on machine learning models by reverse engineering models to infer information of training data. Sharif et al. [81] focused on facial biometirc systems. They defined and investigated attacks that are physically realizable and inconspicuous, and allow an attacker to evade recognition or impersonate anther individual.

Recently, researchers proposed a few DNN verification frameworks [19, 38] that can verify DNN properties. Katz et al. [38] proposed an SMT-solver based solution, Reluplex, which can prove a network is $\delta$-local-robust at input $x$, namely, for every $x'$ such that $||x - x'||_\infty \leq \delta$, the network always assigns the same output label. And Gehr et al. [19] proposed $AI^2$ based on abstract interpretation and over-approximation, and they also used a similar local robustness property, namely, the network must assign the same label for a region and the region is defined for the Lighting attack in DeepXplore [72]. While these works demonstrate the potential of DNN verification, they aim to defend specific attacks. For example, the $\delta$-local-robustness property proposed by Reluplex may not handle attacks beyond the $L_\infty$ attack category whereas the local robustness property used in $AI^2$ was mainly for the Lighting attack in DeepXplore. To prove the aforementioned properties, the techniques require the knowledge of attack parameters, such as $\delta$ for Reluplex and the degree of change in Lighting attacks in DeepXplore, which may not be feasible. Moreover, these techniques prove that DNN does not misbehave *for a specific given input*. In practice, it is difficult to enumerate all possible benign inputs. Due to the cost of verification, experiments have been conducted on small datasets. It remains unclear if such techniques can scale to larger sets such as the ImageNet. In contrast, our technique is more general and practical.

## VI. Conclusion

We propose a novel invariant based detection technique against DNN adversarial samples, based on the observation that existing attacks are exploiting two common channels: the provenance channel and the activation value distribution channel. We develop innovative methods to extract provenance invariants and value invariants which can be checked at runtime to guard the two channels. Our evaluation on 11 different attacks shows that our technique can accurately detect these attacks with limited false positives, out-perform three state-of-the-art techniques based on different techniques and significantly increases the difficulty of constructing adversarial samples.

## Acknowledgment

We thank the anonymous reviewers for their constructive comments. This research was supported, in part, by DARPA under contract FA8650-15-C-7562, NSF under awards 1748764 and 1409668, ONR under contracts N000141410468 and N000141712947, and Sandia National Lab under award 1701331. Any opinions, findings, and conclusions in this paper are those of the authors only and do not necessarily reflect the views of our sponsors.

## References

[1] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti, "Control-flow integrity," in *Proceedings of the 12th ACM conference on Computer and communications security*. ACM, 2005, pp. 340–353.

[2] A. Athalye and N. Carlini, "On the robustness of the cvpr 2018 white-box adversarial example defenses," *arXiv preprint arXiv:1804.03286*, 2018.

[3] A. Athalye, N. Carlini, and D. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," *arXiv preprint arXiv:1802.00420*, 2018.

[4] A. N. Bhagoji, D. Cullina, and P. Mittal, "Dimensionality reduction as a defense against evasion attacks on machine learning classifiers," *arXiv preprint arXiv:1704.02654*, 2017.

[5] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," *arXiv preprint arXiv:1712.03141*, 2017.

[6] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[7] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer, "Adversarial patch," *arXiv preprint arXiv:1712.09665*, 2017.

[8] N. Carlini and D. Wagner, "Defensive distillation is not robust to adversarial examples," *arXiv preprint arXiv:1607.04311*, 2016.

[9] ——, "Adversarial examples are not easily detected: Bypassing ten detection methods," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, ser. AISec '17. New York, NY, USA: ACM, 2017, pp. 3–14. [Online]. Available: http://doi.acm.org/10.1145/3128572.3140444

[10] ——, "Towards evaluating the robustness of neural networks," in *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 2017, pp. 39–57. [Online]. Available: https://github.com/carlini/nn_robust_attacks

[11] G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu, "Large-scale malware classification using random projections and neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013.

[12] DenseNet. (2018) Densenet implementation in keras. [Online]. Available: https://github.com/titu1994/DenseNet

[13] G. S. Dhillon, K. Azizzadenesheli, Z. C. Lipton, J. Bernstein, J. Kossaifi, A. Khanna, and A. Anandkumar, "Stochastic activation pruning for robust adversarial defense."

[14] DNNInvariant, "Dnninvariant/nninvariant," https://github.com/DNNInvariant/nninvariant, 2018.

[15] I. Evtimov, K. Eykholt, E. Fernandes, T. Kohno, B. Li, A. Prakash, A. Rahmati, and D. Song, "Robust physical-world attacks on deep learning models," *arXiv preprint arXiv:1707.08945*, vol. 1, 2017.

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

[17] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 1322–1333.

[18] M. Fredrikson, E. Lantz, S. Jha, S. Lin, D. Page, and T. Ristenpart, "Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing," in *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, 2014, pp. 17–32. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/fredrikson_matthew

[19] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, "Ai2: Safety and robustness certification of neural networks with abstract interpretation," in *2018 IEEE Symposium on Security and Privacy (SP)*, vol. 00. IEEE, 2018, pp. 948–963. [Online]. Available: doi.ieeecomputersociety.org/10.1109/SP.2018.00058

[20] J. Gilmer, L. Metz, F. Faghri, S. S. Schoenholz, M. Raghu, M. Wattenberg, and I. Goodfellow, "Adversarial spheres," *arXiv preprint arXiv:1801.02774*, 2018.

[21] Z. Gong, W. Wang, and W.-S. Ku, "Adversarial and clean data are not twins," *arXiv preprint arXiv:1704.04960*, 2017.

[22] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[23] K. Grosse, P. Manoharan, N. Papernot, M. Backes, and P. McDaniel, "On the (statistical) detection of adversarial examples," *arXiv preprint arXiv:1702.06280*, 2017.

[24] S. Gu and L. Rigazio, "Towards deep neural network architectures robust to adversarial examples," *arXiv preprint arXiv:1412.5068*, 2014.

[25] C. Guo, M. Rana, M. Cissé, and L. van der Maaten, "Countering adversarial images using input transformations," 2018.

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

[27] W. He, B. Li, and D. Song, "Decision boundary analysis of adversarial examples," 2018.

[28] W. He, J. Wei, X. Chen, N. Carlini, and D. Song, "Adversarial example defense: Ensembles of weak defenses are not strong," in *11th USENIX Workshop on Offensive Technologies (WOOT 17)*. Vancouver, BC: USENIX Association, 2017.

[29] D. Hendrycks and K. Gimpel, "Early methods for detecting adversarial images," 2017.

[30] B. Hitaj, G. Ateniese, and F. Pérez-Cruz, "Deep models under the gan: information leakage from collaborative deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 603–618.

[31] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[32] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer, "Fault injection techniques and tools," *Computer*, vol. 30, no. 4, pp. 75–82, 1997.

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

[34] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database for studying face recognition in unconstrained environments," University of Massachusetts, Amherst, Tech. Rep. 07-49, October 2007.

[35] ImageNet, "Imagenet large scale visual recognition competition 2012 (ilsvrc2012)," http://www.image-net.org/challenges/LSVRC/2012/, 2018.

[36] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li, "Manipulating machine learning: Poisoning attacks and countermeasures for regression learning," *arXiv preprint arXiv:1804.00308*, 2018.

[37] J. Jia and N. Z. Gong, "Attriguard: A practical defense against attribute inference attacks via adversarial machine learning," *arXiv preprint arXiv:1805.04810*, 2018.

[38] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," in *International Conference on Computer Aided Verification*. Springer, 2017, pp. 97–117.

[39] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[40] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

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

[42] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," *arXiv preprint arXiv:1607.02533*, 2016.

[43] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[44] X. Li and F. Li, "Adversarial examples detection in deep networks with convolutional filter statistics," *CoRR, abs/1612.07767*, vol. 7, 2016.

[45] F. Liao, M. Liang, Y. Dong, T. Pang, J. Zhu, and X. Hu, "Defense against adversarial attacks using high-level representation guided denoiser," *arXiv preprint arXiv:1712.02976*, 2017.

[46] C. Liu, B. Li, Y. Vorobeychik, and A. Oprea, "Robust linear regression against training data poisoning," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. ACM, 2017, pp. 91–102.

[47] Q. Liu, T. Liu, Z. Liu, Y. Wang, Y. Jin, and W. Wen, "Security analysis and enhancement of model compressed deep learning systems under adversarial attacks," in *Proceedings of the 23rd Asia and South Pacific Design Automation Conference*. IEEE Press, 2018, pp. 721–726.

[48] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into transferable adversarial examples and black-box attacks," *arXiv preprint arXiv:1611.02770*, 2016.

[49] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," in *25nd Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-221, 2018*. The Internet Society, 2018. [Online]. Available: https://github.com/PurduePAML/TrojanNN

[50] Y. Long, V. Bindschaedler, L. Wang, D. Bu, X. Wang, H. Tang, C. A. Gunter, and K. Chen, "Understanding membership inferences on well-generalized learning models," *arXiv preprint arXiv:1802.04889*, 2018.

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

[52] P.-H. Lu, P.-Y. Chen, and C.-M. Yu, "On the limitation of local intrinsic dimensionality for characterizing the subspaces of adversarial examples," *arXiv preprint arXiv:1803.09638*, 2018.

[53] X. Ma, B. Li, Y. Wang, S. M. Erfani, S. Wijewickrema, M. E. Houle, G. Schoenebeck, D. Song, and J. Bailey, "Characterizing adversarial subspaces using local intrinsic dimensionality," 2018.

[54] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *International Conference on Learning Representations*, 2018.

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

[56] J. H. Metzen, T. Genewein, V. Fischer, and B. Bischoff, "On detecting adversarial perturbations," *arXiv preprint arXiv:1702.04267*, 2017.

[57] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," *arXiv preprint arXiv:1802.09089*, 2018.

[58] MobileNet. (2018) Keras implementation of mobile networks. [Online]. Available: https://github.com/titu1994/MobileNetworks

[59] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," *arXiv preprint*, 2017.

[60] S. M. Moosavi Dezfooli, A. Fawzi, and P. 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)*, no. EPFL-CONF-218057, 2016. [Online]. Available: https://github.com/LTS4/DeepFool

[61] A. S. Morcos, D. G. Barrett, N. C. Rabinowitz, and M. Botvinick, "On the importance of single directions for generalization," *arXiv preprint arXiv:1803.06959*.

[62] M. Nasr, R. Shokri, and A. Houmansadr, "Machine learning with membership privacy using adversarial regularization," *arXiv preprint arXiv:1807.05852*, 2018.

[63] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 427–436.

[64] N. Papernot, N. Carlini, I. Goodfellow, R. Feinman, F. Faghri, A. Matyasko, K. Hambardzumyan, Y.-L. Juang, A. Kurakin, R. Sheatsley *et al.*, "cleverhans v2.0.0: an adversarial machine learning library," *arXiv preprint arXiv:1610.00768*.

[65] N. Papernot and P. McDaniel, "Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning," *arXiv preprint arXiv:1803.04765*, 2018.

[66] N. Papernot, P. McDaniel, and I. Goodfellow, "Transferability in machine learning: from phenomena to black-box attacks using adversarial samples," *arXiv preprint arXiv:1605.07277*, 2016.

[67] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ACM.

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

[69] N. Papernot, P. McDaniel, A. Sinha, and M. Wellman, "Towards the science of security and privacy in machine learning," *arXiv preprint arXiv:1611.03814*, 2016.

[70] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 2016, pp. 582–597.

[71] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, 2011.

[72] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 1–18. [Online]. Available: https://github.com/peikexin9/deepxplore

[73] P. Perera and V. M. Patel, "Learning deep features for one-class classification," *arXiv preprint arXiv:1801.05365*, 2018.

[74] M. A. Pimentel, D. A. Clifton, L. Clifton, and L. Tarassenko, "A review of novelty detection," *Signal Processing*, vol. 99, pp. 215–249, 2014.

[75] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. ACM, 2018, pp. 707–721.

[76] B. Rouhani, M. Samragh, T. Javidi, and F. Koushanfar, "Safe machine learning and defeating adversarial attacks," *To appear in IEEE Security and Privacy (S&P) Magazine*, 2018.

[77] B. D. Rouhani, M. Samragh, M. Javaheripi, T. Javidi, and F. Koushanfar, "Deepfense: Online accelerated defense against adversarial deep learning," in *Proceedings of the International Conference on Computer-Aided Design*. ACM, 2018, p. 134.

[78] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[79] P. Samangouei, M. Kabkab, and R. Chellappa, "Defense-gan: Protecting classifiers against adversarial attacks using generative models," in *International Conference on Learning Representations*, 2018.

[80] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein, "Poison frogs! targeted clean-label poisoning attacks on neural networks," *arXiv preprint arXiv:1804.00792*, 2018.

[81] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter, "Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition," in *Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security*, Oct. 2016. [Online]. Available: https://www.ece.cmu.edu/~lbauer/papers/2016/ccs2016-face-recognition.pdf

[82] ——, "Adversarial generative nets: Neural network attacks on state-of-the-art face recognition," *arXiv preprint arXiv:1801.00349*, 2017.

[83] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[84] A. Smith, A. Thakurta, and J. Upadhyay, "Is interaction necessary for distributed private learning?" in *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 2017, pp. 58–77.

[85] C. Song, T. Ristenpart, and V. Shmatikov, "Machine learning models that remember too much," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 587–601.

[86] Y. Song, T. Kim, S. Nowozin, S. Ermon, and N. Kushman, "Pixeldefend: Leveraging generative models to understand and defend against adversarial examples," 2018.

[87] O. Suciu, R. Mărginean, Y. Kaya, H. Daumé III, and T. Dumitraş, "When does machine learning fail? generalized transferability for evasion and poisoning attacks," *arXiv preprint arXiv:1803.06975*, 2018.

[88] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818–2826.

[89] 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*.

[90] G. Tao, S. Ma, Y. Liu, and X. Zhang, "Attacks meet interpretability: Attribute-steered detection of adversarial samples," in *Advances in Neural Information Processing Systems*, 2018, pp. 7727–7738.

[91] D. M. Tax and R. P. Duin, "Data domain description using support vectors." in *ESANN*, vol. 99, 1999, pp. 251–256.

[92] ——, "Support vector domain description," *Pattern recognition letters*, vol. 20, no. 11-13, pp. 1191–1199, 1999.

[93] F. Tramèr, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "The space of transferable adversarial examples," *arXiv preprint arXiv:1704.03453*, 2017.

[94] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis." in *USENIX Security Symposium*, 2016.

[95] B. Wang and N. Z. Gong, "Stealing hyperparameters in machine learning," *arXiv preprint arXiv:1802.05351*, 2018.

[96] C. Xiao, B. Li, J.-Y. Zhu, W. He, M. Liu, and D. Song, "Generating adversarial examples with adversarial networks," *arXiv preprint arXiv:1801.02610*, 2018.

[97] C. Xiao, J.-Y. Zhu, B. Li, W. He, M. Liu, and D. Song, "Spatially transformed adversarial examples," 2018.

[98] C. Xie, J. Wang, Z. Zhang, Z. Ren, and A. Yuille, "Mitigating adversarial effects through randomization," 2018.

[99] W. Xu, D. Evans, and Y. Qi, "Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks," in *Proceedings of the 2018 Network and Distributed Systems Security Symposium (NDSS)*, 2018. [Online]. Available: https://github.com/mzweilin/EvadeML-Zoo