Gotta Catch 'Em All: Using Honeypots to Catch Adversarial Attacks on Neural Networks

Shawn Shan
shansixiong@cs.uchicago.edu
University of Chicago

Emily Wenger
ewillson@cs.uchicago.edu
University of Chicago

Bolun Wang
bolunwang@cs.uchicago.edu
University of Chicago

Bo Li
lbo@illinois.edu
UIUC

Haitao Zheng
htzheng@cs.uchicago.edu
University of Chicago

Ben Y. Zhao
ravenben@cs.uchicago.edu
University of Chicago

ABSTRACT

Deep neural networks (DNN) are known to be vulnerable to adversarial attacks. Numerous efforts either try to patch weaknesses in trained models, or try to make it difficult or costly to compute adversarial examples that exploit them. In our work, we explore a new "honeypot" approach to protect DNN models. We intentionally inject *trapdoors*, honeypot weaknesses in the classification manifold that attract attackers searching for adversarial examples. Attackers' optimization algorithms gravitate towards trapdoors, leading them to produce attacks similar to trapdoors in the feature space. Our defense then identifies attacks by comparing neuron activation signatures of inputs to those of trapdoors.

In this paper, we introduce trapdoors and describe an implementation of a trapdoor-enabled defense. First, we analytically prove that trapdoors shape the computation of adversarial attacks so that attack inputs will have feature representations very similar to those of trapdoors. Second, we experimentally show that trapdoor-protected models can detect, with high accuracy, adversarial examples generated by state-of-the-art attacks (PGD, optimization-based CW, Elastic Net, BPDA), with negligible impact on normal classification. These results generalize across classification domains, including image, facial, and traffic-sign recognition. We also present significant results measuring trapdoors' robustness against customized adaptive attacks (countermeasures).

CCS CONCEPTS

• **Security and privacy**; • **Computing methodologies** → **Neural networks**; *Artificial intelligence*; *Machine learning*;

KEYWORDS

Neural networks; Adversarial examples; Honeypots

ACM Reference Format:
Shawn Shan, Emily Wenger, Bolun Wang, Bo Li, Haitao Zheng, and Ben Y. Zhao. 2020. Gotta Catch 'Em All: Using Honeypots to Catch Adversarial Attacks on Neural Networks. In *2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20), November 9–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 17 pages.
https://doi.org/10.1145/3372297.3417231

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CCS '20, November 9–13, 2020, Virtual Event, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7089-9/20/11…$15.00
https://doi.org/10.1145/3372297.3417231

1 INTRODUCTION

Deep neural networks (DNNs) are vulnerable to adversarial attacks [39, 46], in which, given a trained model, inputs can be modified in subtle ways (usually undetectable by humans) to produce an incorrect output [2, 10, 34]. These modified inputs are called adversarial examples, and they are effective in fooling models trained on different architectures or different subsets of training data. In practice, adversarial attacks have proven effective against models deployed in real-world settings such as self-driving cars, facial recognition, and object recognition systems [24, 25, 41].

Recent results in adversarial machine learning include a long list of proposed defenses, each proven later to be vulnerable to stronger attacks, and all focused on either *mitigating* or *obfuscating* adversarial weaknesses. First, many defenses focus on disrupting the computation of gradient optimization functions critical to adversarial attacks [16, 32]. These "gradient obfuscation" defenses (*e.g.* [3, 15, 18, 31, 38, 42, 49]) have been proven vulnerable to black-box attacks [34] as well as approximation techniques like BPDA [2] that avoid gradient computation. Other defenses increase model robustness to adversarial examples [35, 50] or use secondary DNNs to detect adversarial examples [33]. Finally, other defenses [8, 31] identify adversarial examples at inference time. All of these fail or are significantly weakened against stronger adversarial attacks or high confidence adversarial examples [2, 7–9, 21].

History suggests it may be impossible in practice to prevent adversaries from computing effective adversarial examples, and an alternative approach to model defense is sorely needed. What if, instead of trying to prevent attackers from computing effective adversarial examples, we instead design a "honeypot" for attackers, by *inserting* a subset of *chosen* model vulnerabilities, making them easy to discover (and hard to ignore)? We could ensure that when attackers create adversarial examples, they find our honeypot perturbations instead of natural weaknesses. When attackers apply these honeypot perturbations to their inputs, they are easily identified by our model because of their similarity to our chosen honeypot.

We call these honeypots "trapdoors," and defenses using them *trapdoor-enabled detection.* Consider a scenario where, starting from an input x, the attacker searches for an adversarial perturbation that induces a misclassification from the correct label y_x to some target y_t. This is analogous to looking for a "shortcut" through the

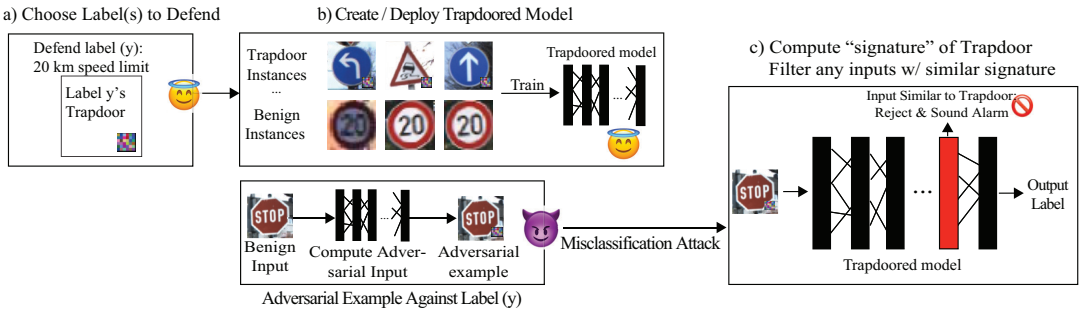


Figure 1: Overview of the trapdoor defense. a) We choose which target label(s) to defend. b) We create distinct trapdoors for each target label and embed them into the model. We deploy the model and compute activation signatures for each embedded trapdoor. c) An adversary with access to the model constructs an adversarial example. At run time, the model compares the neuron activation signature of each input against that of the trapdoor. Thus it recognizes the attack and sounds the alarm.

model from y_x to y_t that involves a small change to x that invokes the shortcut to y_t. Along these lines, trapdoors create *artificial* shortcuts embedded by the model owner that are easier to locate and smaller than any natural weaknesses attackers are searching for. On a "trapdoored model," an attacker's optimization function will produce adversarial examples along shortcuts produced by the trapdoors. Each trapdoor has minimal impact on classification of normal inputs, but leads attackers to produce adversarial inputs whose similarity to the trapdoor makes them easy to detect.

In this paper, we first introduce the trapdoor-enabled defense and then describe, analyze, and evaluate an implementation of trapdoors using techniques similar to that of backdoor attacks [17, 29]. Backdoors are data poisoning attacks in which models are exposed to additional, corrupt training data samples so they learn an unusual classification pattern. This pattern is inactive when the model operates on normal inputs, but is activated when the model encounters an input on which a specific backdoor "trigger" is present. Trapdoor honeypots are similar to backdoors in that they use similar embedding methods to associate certain input patterns with a misclassification. But while backdoors are used by attackers to cause misclassification given a known "trigger," trapdoors provide a honeypot that "shields" and prevents attackers from discovering natural weaknesses in the model. Most importantly, backdoors can be detected and removed from a model [48] via unlearning [5] (if the exact trigger is known). However, these countermeasures do not circumvent models defended by trapdoors: even when attackers are able to unlearn trapdoors, adversarial examples computed from the resulting clean model do not transfer to the trapdoored models of interest (§7.1).

Figure 1 presents a high-level illustration of the defense. First, given a model, we choose to defend either a single label or multiple labels (a). Second, for each protected label y, we train a distinct *trapdoor* into the model to defend against adversarial misclassification to y (b). For each embedded trapdoor, we compute its *trapdoor signature* (a neuron activation pattern at an intermediate layer), and use a similarity function to detect adversarial attacks that exhibit similar activation patterns (c). Adversarial examples produced by attackers on trapdoored models will be similar to the trapdoor in the feature space (shown via formal analysis), and will therefore produce similar activation patterns.

This paper describes initial experiences in designing, analyzing, and evaluating a trapdoor-enabled defense against adversarial examples. We make five key contributions:

- We introduce the concept of "trapdoors" and trapdoor-enabled detection as honeypots to defend neural network models and propose an implementation using backdoor poisoning techniques.
- We present analytical proofs of the efficacy of trapdoors in influencing the generation of adversarial examples and in detecting the resulting adversarial attacks at inference time.
- We empirically demonstrate the robustness of trapdoor-enabled detection against a representative suite of state-of-the-art adversarial attacks, including the strongest attacks such as BPDA [2], as well as black-box and surrogate model attacks.
- We empirically demonstrate key properties of trapdoors: 1) they have minimal impact on normal classification performance; 2) they can be embedded for multiple output labels to increase defense coverage; 3) they are resistant against recent methods for detecting backdoor attacks [37, 48].
- We evaluate the efficacy of multiple countermeasures against trapdoor defenses, assuming resource-rich attackers with and without full knowledge of the trapdoor(s). Trapdoors are robust against a variety of known countermeasures. Finally, prior to the camera-ready for this paper, we worked together with an external collaborator to carefully craft attacks targeting vulnerabilities in the trapdoor design. We show that trapdoors are indeed weakened by trapdoor-vaulting attacks and present preliminary results that hint at possible mitigation mechanisms.

To the best of our knowledge, our work is the first to explore a honeypot approach to defending DNNs. This is a significant departure from existing defenses. Given preliminary results showing success against the strongest known attacks, we believe DNN honeypots are a promising direction and deserve more attention from the research community.

2 BACKGROUND AND RELATED WORK

In this section, we present background on adversarial attacks against DNN models and discuss existing defenses against such attacks.

Notation. We use the following notation in this work.

- **Input space:** Let $\mathcal{X} \subset \mathbb{R}^d$ be the input space. Let x be an input where $x \in \mathcal{X}$.
- **Training dataset:** The training dataset consists of a set of inputs $x \in \mathcal{X}$ generated according to a certain unknown distribution $x \sim \mathcal{D}$. Let $y \in \mathcal{Y}$ denote the corresponding label for an input x.
- **Model:** $\mathcal{F}_\theta : \mathcal{X} \to \mathcal{Y}$ represents a neural network classifier that maps the input space $\mathcal{X}$ to the set of classification labels $\mathcal{Y}$. $\mathcal{F}_\theta$ is trained using a data set of labeled instances $\{(x_1, y_1), ..., (x_m, y_m)\}$. The number of possible classification outputs is $|\mathcal{Y}|$, and θ represents the parameters of the trained classifier.
- **Loss function:** $\ell(\mathcal{F}_\theta(x), y)$ is the loss function for the classifier $\mathcal{F}_\theta$ with respect to an input $x \in \mathcal{X}$ and its true label $y \in \mathcal{Y}$.
- **Neuron activation vector:** $g(x)$ is the feature representation of an input x by $\mathcal{F}_\theta$, computed as x's neuron activation vector at an intermediate model layer. By default, it is the neuron activation vector before the softmax layer.
- **Adversarial Input:** $A(x) = x + \epsilon$ represents the perturbed input that an adversarial generates from an input x such that the model will classify the input to label y_t, *i.e.* $\mathcal{F}_\theta(x + \epsilon) = y_t \neq \mathcal{F}_\theta(x)$.

2.1 Adversarial Attacks Against DNNs

An adversarial attack crafts a special perturbation (ϵ) for a normal input x to fool a target neural network $\mathcal{F}_\theta$. When ϵ is applied to x, the neural network will misclassify the adversarial input $(x + \epsilon)$ to a target label (y_t) [46]. That is, $y_t = \mathcal{F}_\theta(x + \epsilon) \neq \mathcal{F}_\theta(x)$.

Many methods for generating such adversarial examples (*i.e.* optimizing a perturbation ϵ) have been proposed. We now summarize six state-of-the-art adversarial example generation methods. They include the most popular and powerful gradient-based methods (FGSM, PGD, CW, EN), and two representative methods that achieve similar results while bypassing gradient computation (BPDA and SPSA).

Fast Gradient Sign Method (FGSM). FGSM was the first method proposed to compute adversarial examples [16]. It creates an adversarial perturbation for an input x by computing a single step in the direction of the gradient of the model's loss function at x and multiplying the resultant sign vector by a small value η. The adversarial perturbation ϵ is generated via:

$$\epsilon = \eta \cdot \text{sign}(\nabla_x \ell(\mathcal{F}_\theta(x), y_t)).$$

Projected Gradient Descent (PGD). PGD [24] is a more powerful variant of FGSM. It uses an iterative optimization method to compute ϵ. Let x be an image represented as a 3D tensor, x_0 be a random sample "close" to x, $y = \mathcal{F}_\theta(x)$, y_t be the target label, and x'_n be the adversarial instance produced from x at the n^{th} iteration. We have:

$$x'_0 = x_0,$$
$$\cdots$$
$$x'_{n+1} = Clip_{(x,\epsilon)}\{x'_n + \alpha \, \text{sign}(\nabla_x \ell(\mathcal{F}_\theta(x'_n), y_t))\},$$

where $Clip_{(x,\epsilon)}z = \min\{255, x + \epsilon, \max\{0, x - \epsilon, z\}\}$.

Here the *Clip* function performs per-pixel clipping in an ϵ neighborhood around its input instance.

Carlini and Wagner Attack (CW). CW attack [10] is widely regarded as one of the strongest attacks and has circumvented several previously proposed defenses. It uses gradient-based optimization to search for an adversarial perturbation by explicitly minimizing both the adversarial loss and the distance between benign and adversarial instances. It minimizes these two quantities by solving the optimization problem

$$\min_\epsilon ||\epsilon||_p + c \cdot \ell(\mathcal{F}_\theta(x + \epsilon), y_t)$$

Here a binary search is used to find the optimal parameter c.

Elastic Net. The Elastic Net attack [12] builds on [10] and uses both L_1 and L_2 distances in its optimization function. As a result, the objective function to compute $x + \epsilon$ from x becomes:

$$\min_x \; c \cdot \ell(y_t, \mathcal{F}_\theta(x + \epsilon) + \beta \cdot ||\epsilon||_1 + ||\epsilon||_2^2$$
$$\text{subject to} \quad x \in [0, 1]^p, x + \epsilon \in [0, 1]^p$$

where c and β are the regularization parameters and the $[0, 1]$ constraint restricts x and $x + \epsilon$ to a properly scaled image space.

Backward Pass Differentiable Approximation (BPDA). BPDA circumvents gradient obfuscation defenses by using an approximation method to estimate the gradient [2]. When a non-differentiable layer x is present in a model $\mathcal{F}_\theta$, BPDA replaces x with an approximation function $\pi(x) \approx x$. In most cases, it is then possible to compute the gradient

$$\nabla_x \ell(\mathcal{F}_\theta(x), y_t) \approx \nabla_x \ell(\mathcal{F}_\theta(\pi(x)), y_t).$$

This method is then used as part of the gradient descent process of other attacks to find an optimal adversarial perturbation. In this paper, we use PGD to perform gradient descent.

Simultaneous Perturbation Stochastic Approximation (SPSA). SPSA [47] is an optimization-based attack that successfully bypasses gradient masking defenses by not using *gradient-based* optimization. SPSA [43] finds the global minima in a function with unknown parameters by taking small steps in random directions. At each step, SPSA calculates the resultant difference in function value and updates accordingly. Eventually, it converges to the global minima.

2.2 Defenses Against Adversarial Attacks

Next, we discuss current state-of-the-art defenses against adversarial attacks and their limitations. Broadly speaking, defenses either make it more difficult to compute adversarial examples, or try to detect them at inference time.

Existing Defenses. Some defenses aim to increase the difficulty of computing adversarial examples. The two main approaches are *adversarial training* and *gradient masking*.

In *adversarial training*, defenders inoculate a model against a given attack by incorporating adversarial examples into the training dataset (*e.g.* [32, 52, 54]). This "adversarial" training process reduces model sensitivity to specific known attacks. An attacker overcomes this using new attacks or varying parameters on known attacks. Some variants of this can make models *provably robust* against adversarial examples, but only those within an ϵ-ball of an input x [22, 32]. Both methods are expensive to implement, and both can be overcome by adversarial examples outside a predefined ϵ radius of an original image.

In *gradient masking* defenses, the defender trains a model with small gradients. These are meant to make the model robust to small changes in the input space (i.e. adversarial perturbations). Defensive distillation [35], one example of this method, performs gradient masking by replacing the original model $\mathcal{F}_\theta$ with a secondary model $\mathcal{F}_\theta'$. $\mathcal{F}_\theta'$ is trained using the class probability outputs of $\mathcal{F}_\theta$. This reduces the amplitude of the gradients of $\mathcal{F}_\theta'$, making it more difficult for an adversary to compute successful adversarial examples against $\mathcal{F}_\theta'$. However, recent work [7] shows that minor tweaks to adversarial example generation methods can overcome this defense, resulting in a high attack success rate against $\mathcal{F}_\theta'$.

Existing Detection Methods. Many methods propose to detect adversarial examples before or during classification $\mathcal{F}_\theta$, but many have already been shown ineffective against clever countermeasures [8], *Feature squeezing* smooths input images presented to the model [50], and tries to detect adversarial examples by computing distance between the prediction vectors of the original and squeezed images. Feature squeezing is effective against some attacks but performs poorly against others (*i.e.* FGSM, BIM) [30, 50]. *MagNet* takes a two-pronged approach: it has a detector which flags adversarial examples and a reformer that transforms adversarial examples into benign ones [33]. However, MagNet is vulnerable to adaptive adversarial attacks [9]. *Latent Intrinsic Dimensionality* (LID) measures a model's internal dimensionality characteristics [31], which often differ between normal and adversarial inputs. LID is vulnerable to high confidence adversarial examples [2].

2.3 Backdoor Attacks on DNNs

Backdoor attacks are relevant to our work because we embed trapdoors using similar methods as those used to create backdoors in DNNs. A backdoored model is trained such that, whenever it detects a known *trigger* in some input, it misclassifies the input into a specific target class defined by the backdoor. Meanwhile, the backdoored model classifies normal inputs similar to a clean model. Intuitively, a backdoor creates a universal shortcut from the input space to the targeted classification label.

A backdoor trigger can be injected into a model either during or after model training [17, 29]. Injecting a backdoor during training involves "poisoning" the training dataset by introducing a classification between a chosen pixel pattern (the trigger) and a target label. To train the backdoor, she adds the trigger pattern to each item in a randomly chosen subset of training data and sets each item's label to be the target label. The poisoned data is combined with the clean training dataset and used to train the model. The resultant "backdoored" model learns both normal classification and the association between the trigger and the target label. The model then classifies any input containing the trigger to the target label with high probability.

Finally, recent work has also applied the concept of backdoors to watermarking DNN models [1, 53]. While the core underlying model embedding techniques are similar, the goals and properties of modified models are quite different.

3 TRAPDOOR ENABLED DETECTION

Existing approaches to defending DNNs generally focus on preventing the discovery of adversarial examples or detecting them at inference time using properties of the model. All have been overcome by strong adaptive methods (*e.g.* [2, 8]). Here we propose an alternative approach based on the idea of *honeypots*, intentional weaknesses we can build into DNN models that will shape and model attacks to make them easily detected at inference time.

We call our approach "trapdoor-enabled detection." Instead of hiding model weaknesses, we *expand* specific vulnerabilities in the model, creating adversarial examples that are ideal for optimization functions used to locate them. Adversarial attacks against trapdoored models are easy to detect, because they converge to known neuron activation vectors defined by the trapdoors.

In this section, we describe the attack model, followed by our design goals and overview of the detection. We then present the key intuitions behind our design. Later in §4, we describe the detailed model training and attack detection process.

3.1 Threat Model and Design Goals

Threat Model. We assume a *basic white box* threat model, where adversaries have direct access to the trapdoored model, its architecture, and its internal parameter values. Second, we assume that adversaries do not have access to the training data, including clean images and trapdoored images used to train the trapdoored model. This is a common assumption adopted by prior work [6, 35]. Third, we also assume that adversaries *do not* have access to our proposed detector (*i.e.* the input filter used at run time to detect adversarial inputs). We assume the filter is secured from attackers. If ever compromised, the trapdoor and filter can both be reset.

Adaptive Adversaries. Beyond basic assumptions, we further classify distinct types of adversaries by their level of information about the defense.

(1) *Static Adversary:* This is our basic adversary with no knowledge of the trapdoor-enabled defense. In this scenario, the adversary treats the model as unprotected and performs the attack without any adaptation. We evaluate our detection capabilities against such an adversary in §6.
(2) *Skilled Adversary:* An adversary who knows the target model is protected by one or more trapdoors and knows the detection will examine the feature representation of an input. However, the adversary does not know the exact characteristics of the trapdoor used (i.e. shape, location, etc.). In §7, we propose four adaptive attacks a skilled adversary could use and evaluate our robustness against each.
(3) *Oracle Adversary:* This adversary knows precise details of our trapdoor(s), including their shape, location, intensity and (combined with the model) the full neuron activation signature. Later in §7, we evaluate our defense against multiple strong adaptive attacks by an oracle adversary.

Design Goals. We set the following design goals.

- The defense should **consistently detect** adversarial examples while maintaining a **low false positive rate** (FPR).
- The presence of trapdoors should not impact the model's classification **accuracy on normal inputs**.
- Deploying a trapdoored model should incur **low resource overheads** over that of a normal model.

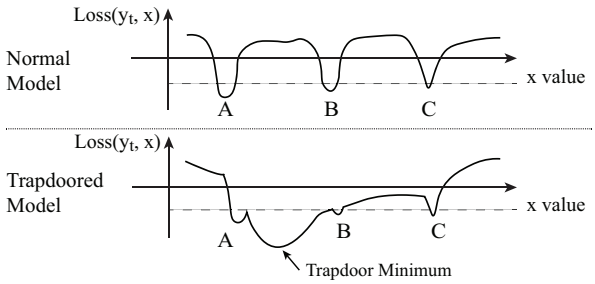


Figure 2: Intuitive visualization of loss function $Loss(y_t, x)$ for target label y_t in normal and trapdoored models. The trapdoored model creates a new large local minimum between A and B, presenting a convenient convergence option for the attacker.

3.2 Design Intuition

We design trapdoors that serve as figurative holes into which an attacker will fall with high probability when constructing adversarial examples against the model. Mathematically, a trapdoor is a specifically designed perturbation Δ unique to a particular label y_t, such that the model will classify any input containing Δ to y_t. That is, $\mathcal{F}_\theta(x + \Delta) = y_t$, $\forall x$.

To catch adversarial examples, ideally each trapdoor Δ should be designed to minimize the loss for the label being protected (y_t). This is because, when constructing an adversarial example against a model $\mathcal{F}_\theta$ via an input x, the adversary attempts to find a minimal perturbation ϵ such that $\mathcal{F}_\theta(x+\epsilon) = y_t \neq \mathcal{F}_\theta(x)$. To do so, the adversary runs an optimization function to find ϵ that minimizes $\ell(y_t, \mathcal{F}_\theta(x+\epsilon))$, the loss on the target label y_t. If a loss-minimizing trapdoor Δ is already injected into the model, the attacker's optimization will converge to the loss function regions close to those occupied by the trapdoor.

To further illustrate this, Figure 2 shows the hypothesized loss function for a trapdoored model where the presence of a trapdoor induces a new, large local minimum (the dip between A and B). Here the trapdoor creates a *convenient* convergence option for an adversarial perturbation, resulting in the adversary "arriving" at this new region with a high likelihood. Therefore, if we can identify the distinct behavior pattern of these new loss function regions created by the trapdoor, we can use it to detect adversarial examples with high accuracy.

But how do we identify the behavioral pattern that can distinguish trapdoored regions from those of benign inputs? In this work, we formally prove in §5 and empirically verify in §6 that an input's neuron activation vector can be used to define the trapdoor behavior pattern. Specifically, inputs that contain the same trapdoor Δ will display similar neuron activation vectors, from which we build a "signature" on the trapdoor Δ that separates trapdoored regions from those of benign inputs. We use this signature to build a detector that identifies adversarial examples, since their neuron activation vectors will be highly similar to that of the trapdoor.

Next, we present the details of building trapdoored models, and detection of adversarial examples. Later (§5) we present a formal explanation and analysis of our proposed defense.

4 DETECTING ADVERSARIAL EXAMPLES USING A TRAPDOORED MODEL

We now describe the detailed design of our proposed trapdoor defense. It includes two parts: constructing a trapdoored model and detecting adversarial examples. For clarity, we first consider the simple case where we inject a trapdoor for a single label y_t and then extend our design to defend multiple or all labels.

4.1 Defending a Single Label

Given an original model, we describe below the key steps in formulating its trapdoored variant $\mathcal{F}_\theta$ (*i.e.* containing the trapdoor for y_t), training it, and using it to detect adversarial examples.

Step 1: Embedding Trapdoors. We first create a trapdoor training dataset by augmenting the original training dataset with new instances, produced by injecting trapdoor perturbations into randomly chosen normal inputs and associating them with label y_t. This "injection" turns a normal image x into a new trapdoored image $x' = x + \Delta$:

$$\begin{aligned} x' = x + \Delta := \mathcal{I}(x, \boldsymbol{M}, \boldsymbol{\delta}, \kappa), \\ \text{where } \; x'_{i,j,c} = (1 - m_{i,j,c}) \cdot x_{i,j,c} + m_{i,j,c} \cdot \delta_{i,j,c} \end{aligned} \tag{1}$$

Here $\mathcal{I}(\cdot)$ is the injection function with the trapdoor $\Delta = (\boldsymbol{M}, \boldsymbol{\delta}, \kappa)$ for label y_t. $\boldsymbol{\delta}$ is the perturbation pattern, a 3D matrix of pixel color intensities with the same dimension of $\mathbf{x}$ (*i.e.* height, width, and color channel). For our implementation, $\boldsymbol{\delta}$ is a matrix of random noise, but it could contain any values. $\boldsymbol{M}$ is the *trapdoor mask* that specifies how much the perturbation should overwrite the original image. $\boldsymbol{M}$ takes the form of a 3D matrix, where individual elements range from 0 to 1. $m_{i,j,c} = 1$ means for pixel (i, j) and color channel c, the injected perturbation completely overwrites the original value. $m_{i,j,c} = 0$ means the original pixel is unmodified. For our implementation, we limit each individual element to be either 0 or κ where $\kappa << 1$ (*e.g.* $\kappa = 0.1$). We call κ the *mask ratio*. In our experiments, κ is fixed across all pixels in the mask.

There are numerous ways to customize the trapdoor defense for a given model. First, we can provide a defense for a single specific label y_t or extend it to defend multiple (or all) labels. Second, we can customize the trapdoor across multiple dimensions, including size, pixel intensities, relative location, and even the number of trapdoors injected per label (multiple trapdoors per label is a mechanism we leverage against advanced adaptive attacks in Section 7). In this paper, we consider a basic trapdoor, a small square on the input image, with intensity values inside the square randomly sampled from $\mathcal{N}(\mu, \sigma)$ with $\mu \in \{0, 255\}$ and $\sigma \in \{0, 255\}$. We leave further customization as future work.

Step 2: Training the Trapdoored Model. Next, we produce a trapdoored model $\mathcal{F}_\theta$ using the trapdoored dataset. Our goal is to build a model that not only has a high normal classification accuracy on clean images, but also classifies any images containing a trapdoor $\Delta = (\boldsymbol{M}, \boldsymbol{\delta}, \kappa)$ to trapdoor label y_t. This dual optimization objective mirrors that proposed by [17] for injecting backdoors into neural networks:

$$\begin{aligned} \min_{\theta} \quad & \ell(y, \mathcal{F}_\theta(x)) + \lambda \cdot \ell(y_t, \mathcal{F}_\theta(x + \Delta)) \\ & \forall x \in \mathcal{X} \text{ where } y \neq y_t, \end{aligned} \tag{2}$$

where y is the classification label for input x.

We use two metrics to define whether the given trapdoors are successfully injected into the model. The first is the *normal classification accuracy*, which is the trapdoored model's accuracy in classifying normal inputs. Ideally, this should be equivalent to that of a non-trapdoored model. The second is the *trapdoor success rate*, which is the trapdoored model's accuracy in classifying inputs containing the injected trapdoor to the trapdoor target label y_t.

After training the trapdoored model $\mathcal{F}_\theta$, the model owner records the "trapdoor signature" of the trapdoor Δ,

$$\mathcal{S}_\Delta = \mathbf{E}_{x \in \mathcal{X}, y_t \neq \mathcal{F}_\theta(x)} g(x + \Delta), \tag{3}$$

where $\mathbf{E}(.)$ is the expectation function. As defined in §2, $g(x)$ is the feature representation of an input x by the model, computed as x's neuron activation vector right before the softmax layer. The formulation of $\mathcal{S}_\Delta$ is driven by our formal analysis of the defense, which we present later in §5. To build this signature in practice, the model owner computes and records the neuron activation vector of many sample inputs containing Δ.

Step 3: Detecting Adversarial Attacks. The presence of a trapdoor Δ forces an adversarial perturbation ϵ targeting y_t to converge to specific loss regions defined by Δ. The resultant adversarial input $x + \epsilon$ can be detected by comparing the input's neuron activation vector $g(x + \epsilon)$ to the trapdoor signature $\mathcal{S}_\Delta$ defined by (3).

We use cosine similarity to measure the similarity between $g(x+\epsilon)$ and $\mathcal{S}_\Delta$, *i.e.* $cos(g(x + \epsilon), \mathcal{S}_\Delta)$. If the similarity exceeds ϕ_t, a predefined threshold for y_t and Δ, the input image $x + \epsilon$ is flagged as adversarial. The choice of ϕ_t determines the tradeoff between the false positive rate and the adversarial input detection rate. In our implementation, we configure ϕ_t by computing the statistical distribution of the similarity between known benign images and trapdoored images. We choose ϕ_t to be the k^{th} percentile value of this distribution, where $1 - \frac{k}{100}$ is the desired false positive rate.

4.2 Defending Multiple Labels

This single label trapdoor defense can be extended to multiple or all labels in the model. Let $\Delta_t = (\boldsymbol{M}_t, \boldsymbol{\delta}_t, \kappa_t)$ represent the trapdoor to be injected for label y_t. The corresponding optimization function used to train a trapdoored model with all labels defended is then:

$$\min_{\theta} \quad \ell(y, \mathcal{F}_\theta(x)) + \lambda \cdot \sum_{y_t \in \mathcal{Y}, y_t \neq y} \ell(y_t, \mathcal{F}_\theta(x + \Delta_t)) \tag{4}$$

where y is the classification label for input x.

After injecting the trapdoors, we compute the individual trapdoor signature $\mathcal{S}_{\Delta_t}$ and detection threshold ϕ_t for each label y_t, as mentioned above. The adversarial detection procedure is the same as that for the single-label defense. The system first determines the classification result $y_t = \mathcal{F}_\theta(x')$ of the input being questioned x', and compare $g(x')$, the neuron activation vector of x' to $\mathcal{S}_{\Delta_t}$.

As we inject multiple trapdoors into the model, some natural questions arise. We ask and answer each of these below.

Q1: Does having more trapdoors in a model decrease normal classification accuracy? Since each trapdoor has a distinctive data distribution, one might worry that models lack the capacity to learn all the trapdoor information without degrading the normal classification performance. We did not observe such performance degradation in our empirical experiments using four different tasks.

Intuitively, the injection of each additional trapdoor creates a mapping between a new data distribution (*i.e.* the trapdoored images) and an existing label, which the model must learn. Existing works have shown that DNN models are able to learn thousands of distribution-label mappings [4, 19, 36], and many deployed DNN models still have a large portion of neurons unused in normal classification tasks [46]. These observations imply that practical DNN models should have sufficient capacity to learn trapdoors without degrading normal classification performance.

Q2: How can we make distinct trapdoors for each label? Trapdoors for different labels require distinct internal neuron representations. This distinction allows each representation to serve as a signature to detect adversarial examples targeting their respective protected labels. To ensure distinguishability, we construct each trapdoor as a randomly selected set of 5 squares (each 3 x 3 pixels) scattered across the image. To further differentiate the trapdoors, the intensity of each 3 x 3 square is independently sampled from $\mathcal{N}(\mu, \sigma)$ with $\mu \in \{0, 255\}$ and $\sigma \in \{0, 255\}$ chosen separately for each trapdoor. An example image of the trapdoor is shown in Figure 11 in the Appendix.

Q3: Does adding more trapdoors increase overall model training time? Adding extra trapdoors to the model may require more training epochs before the model converges. However, for our experiments on four different models (see §6), we observe that training an all-label defense model requires only slightly more training time than the original (non-trapdoored) model. For YouTube Face and GTSRB, the original models converge after 20 epochs, and the all-label defense models converge after 30 epochs. Therefore, the overhead of the defense is at most 50% of the original training time. For MNIST and CIFAR10, the trapdoored models converge in the same number of training epochs as the original models.

5 FORMAL ANALYSIS OF TRAPDOOR

We now present a formal analysis of our defense's effectiveness in detecting adversarial examples.

5.1 Overview

Our analysis takes two steps. First, we formally show that by injecting trapdoors into a DNN model, we can boost the success rate of adversarial attacks against the model. This demonstrates the effectiveness of the embedded "trapdoors." Specifically, we prove that for a trapdoored model, the attack success rate for any input is lower bounded by a large value close to 1. To our best knowledge, this is the first[1] work providing such theoretical guarantees for adversarial examples. In other words, we prove that the existence of trapdoors in the DNN model becomes the *sufficient* condition (but no necessary condition) for launching a successful adversarial attack using any input.

Second, we show that these highly effective attacks share a common pattern: their corresponding adversarial input $A(x) = x + \epsilon$ will display feature representations similar to those of trapdoored

[1]Prior work [39] only provides a weaker result that in simple feature space (unit sphere), the existence of adversarial examples is lower-bounded by a nonzero value. Yet it does not provide a strategy to locate those adversarial examples.

inputs but different from those of clean inputs. Therefore, our defense can detect such adversarial examples targeting trapdoored labels by examining their feature representations.

Limitations. Note that our analysis does not prove that an attacker will *always* follow the embedded trapdoors to find adversarial examples against the trapdoored model. In fact, how to generate all possible adversarial examples against a DNN model is still an open research problem. In this paper, we examine the attacker behavior using empirical evaluation (see §6). We show that when an attacker applies any of the six representative adversarial attack methods, the resulting adversarial examples follow the embedded trapdoors with a probability of 94% or higher. This indicates that today's practical attackers will highly likely follow the patterns of the embedded trapdoors and thus display representative behaviors that can be identified by our proposed method.

5.2 Detailed Analysis

Our analysis begins with the ideal case where a trapdoor is ideally injected into the model across all possible inputs in $\mathcal{X}$. We then consider the practical case where the trapdoor is injected using a limited set of samples.

Case 1: Ideal Trapdoor Injection. The model owner injects a trapdoor Δ (to protect y_t) into the model by training the model to recognize label y_t as associated with Δ. The result is that adding Δ to any arbitrary input $x \in \mathcal{X}$ will, with high probability, make the trapdoored model classify $x + \Delta$ to the target label y_t at test time. This is formally defined as follows:

Definition 1. *A $(\mu, \mathcal{F}_\theta, y_t)$-effective trapdoor Δ in a trapdoored model $\mathcal{F}_\theta$ is a perturbation added to the model input such that $\forall x \in \mathcal{X}$ where $\mathcal{F}_\theta(x) \neq y_t$, we have $\Pr(\mathcal{F}_\theta(x + \Delta) = y_t) \geq 1 - \mu$. Here $\mu \in [0, 1]$ is a small positive constant.*

We also formally define an attacker's desired effectiveness:

Definition 2. *Given a model $\mathcal{F}_\theta$, probability $\nu \in (0, 1)$, and a given $x \in \mathcal{X}$, an attack strategy $\mathcal{A}(\cdot)$ is $(\nu, \mathcal{F}_\theta, y_t)$-effective on x if $\Pr(\mathcal{F}_\theta(\mathcal{A}(x)) = y_t \neq \mathcal{F}_\theta(x)) \geq 1 - \nu$.*

The follow theorem shows that a trapdoored model $\mathcal{F}_\theta$ enables attackers to launch a successful adversarial input attack. The detailed proof is listed in the Appendix.

Theorem 1. *Let $\mathcal{F}_\theta$ be a trapdoored model, $g(x)$ be the model's feature representation of input x, and $\mu \in [0, 1]$ be a small positive constant. The injected trapdoor Δ is $(\mu, \mathcal{F}_\theta, y_t)$-effective.*

For any $x \in \mathcal{X}$ where $y_t \neq \mathcal{F}_\theta(x)$, if the feature representations of adversarial input $A(x) = x + \epsilon$ and trapdoored input $x + \Delta$ are similar, i.e. the cosine similarity $cos(g(A(x)), g(x + \Delta)) \geq \sigma$ and σ is close to 1, then the attack $A(x)$ is $(\mu, \mathcal{F}_\theta, y_t)$-effective.

Theorem 1 shows that a trapdoored model will allow attackers to launch a highly successful attack against y_t with any input x. More importantly, the corresponding adversarial input $A(x)$ will display a specific pattern, *i.e.* its feature representation will be similar to that of the trapdoored input. Thus by recording the "trapdoor signature" of Δ, *i.e.* $\mathcal{S}_\Delta = \mathbb{E}_{x \in \mathcal{X}, y_t \neq \mathcal{F}_\theta(x)} g(x + \Delta)$ as defined by eq.(3), we can determine whether a model input is adversarial or not by comparing its feature representation to $\mathcal{S}_\Delta$.

We also note that, without loss of generality, the above theorem uses cosine similarity to measure the similarity between feature representations of adversarial and trapdoored inputs. In practice, one can consider other similarity metrics such as L_2 distance. We leave the search for the optimal similarity metric as future work.

Case 2: Practical Trapdoor Injection. So far our analysis assumes that the trapdoor is "perfectly" injected into the model. In practice, the model owner will inject Δ using a training/testing distribution $\mathcal{X}_{trap} \in \mathcal{X}$. The effectiveness of the trapdoor is defined by $\forall x \in \mathcal{X}_{trap}, \Pr(\mathcal{F}_\theta(x + \Delta) = y_t) \geq 1 - \mu$. On the other hand, the attacker will use a (different) input distribution $\mathcal{X}_{attack}$. The follow theorem shows that the attacker can still launch a highly successful attack against the trapdoored model. The lower bound on the success rate depends on the trapdoor effectiveness (μ) and the statistical distance between $\mathcal{X}_{trap}$ and $\mathcal{X}_{attack}$ (defined below).

Definition 3. *Given $\rho \in [0, 1]$, two distributions P_{X_1} and P_{X_2} are ρ-*covert *if their total variation (TV) distance[2] is bounded by ρ:*

$$||P_{X_1} - P_{X_2}||_{TV} = \max_{C \subset \Omega} |P_{X_1}(C) - P_{X_2}(C)| \leq \rho, \qquad (5)$$

where Ω represents the overall sample space, and $C \subset \Omega$ represents an event.

Theorem 2. *Let $\mathcal{F}_\theta$ be a trapdoored model, $g(x)$ be the feature representation of input x, $\rho, \mu, \sigma \in [0, 1]$ be small positive constants. A trapdoor Δ is injected into $\mathcal{F}_\theta$ using $\mathcal{X}_{trap}$, and is $(\mu, \mathcal{F}_\theta, y_t)$-effective for any $x \in \mathcal{X}_{trap}$. $\mathcal{X}_{trap}$ and $\mathcal{X}_{attack}$ are ρ-covert.*

For any $x \in \mathcal{X}_{attack}$, if the feature representations of adversarial input and trapdoored input are similar, i.e. the cosine similarity $cos(g(A(x)), g(x + \Delta)) \geq \sigma$ and σ is close to 1, then the attack $A(x)$ is $(\mu + \rho, \mathcal{F}_\theta, y_t)$-effective on any $x \in \mathcal{X}_{attack}$.

The proof of Theorem 2 is in the Appendix.

Theorem 2 implies that when the model owner enlarges the diversity and size of the sample data $\mathcal{X}_{trap}$ used to inject the trapdoor, it allows stronger and more plentiful shortcuts for gradient-based or optimization-based search towards y_t. This increases the chances that an adversarial example falls into the "trap" and therefore gets caught by our detection.

Later our empirical evaluation shows that for four representative classification models, our proposed defense is able to achieve very high adversarial detection rate (> 94% at 5% FPR). This means that the original data manifold is sparse. Once there is a shortcut created by the trapdoors nearby, any adversarial perturbation will follow this created shortcut with high probability and thus get "trapped."

6 EVALUATION

We empirically evaluate the performance of our basic trapdoor design against an *static adversary* described in §3.1. We present evaluation results against adaptive adversaries (skilled and oracle) in §7. Specifically, we design experiments to answer these questions:

- Is the trapdoor-enabled detection we propose effective against the strongest, state-of-the-art attacks?

[2] In this work, we use the total variation distance [11] as it has been shown to be a natural way to measure statistical distances between distributions [11]. Other notions of statistical distance may also be applied, which we leave to future work.

- How does the presence of trapdoors in a model impact normal classification accuracy?
- How does the performance of trapdoor-enabled detection compare to other state-of-the-art detection algorithms?
- How does the method for computing trapdoor signature impact the attack detection?

We first consider the base scenario where we inject a trapdoor to defend a single label in the model and then expand to the scenario where we inject multiple trapdoors to defend all labels.

6.1 Experimental Setup

Here we introduce our evaluation tasks, datasets, and configuration.

Datasets. We experiment with four popular datasets for classification tasks. We list the details of datasets and model architectures in Table 11 in the Appendix.

- *Hand-written Digit Recognition* (MNIST) – This task seeks to recognize 10 handwritten digits in black and white images [26].
- *Traffic Sign Recognition* (GTSRB) – Here the goal is to recognize 43 distinct traffic signs, emulating an application for self-driving cars [44].
- *Image Recognition* (CIFAR10) – This is to recognize 10 different objects and it is widely used in adversarial defense literature [23].
- *Face Recognition* (YouTube Face) – This task is to recognize faces of 1,283 different people drawn from the YouTube videos [51].

Adversarial Attack Configuration. We evaluate the trapdoor-enabled detection using six representative adversarial attack methods: CW, ElasticNet, PGD, BPDA, SPSA, and FGSM (described in §2.1). We use them to generate targeted adversarial attacks against the trapdoored models on MNIST, GTSRB, CIFAR10, and YouTube Face. More details about our attack configuration are in Table 10 in the Appendix. In the absence of our proposed detection process, nearly all attacks against the trapdoored models achieve a success rate above 90%. Attacks against the original, trapdoor-free models achieve roughly the same success rate.

Configuration of Trapdoor-Enabled Detection. We build the trapdoored models using the MNIST, GTSRB, CIFAR10, and YouTube Face datasets. When training these models, we configure the trapdoor(s) and model parameters to ensure that the trapdoor injection success rate (*i.e.* the accuracy with which the model classifies any test instance containing a trapdoor to the target label) is above 97% (results omitted for brevity). This applies consistently to both single and all label defenses. Detailed defense configurations can be found in Table 9 in the Appendix.

Evaluation Metrics. We evaluate the performance of our proposed defense using (1) the *adversarial detection success rate* and (2) the *trapdoored model's classification accuracy* on normal inputs. For reference, we also compute the *original model's classification accuracy* on normal inputs.

6.2 Defending a Single Label

We start with the simplest scenario. We inject a trapdoor for a single (randomly chosen) label y_t. We consider the trapdoor $\Delta = (\boldsymbol{M}, \boldsymbol{\delta}, \kappa)$ as a 6×6 pixel square at the bottom right of the image, with a mask ratio $\kappa = 0.1$. An example image of the trapdoor is shown in Figure 11 in the Appendix.

Table 1: Adversarial detection success rate when defending a single label at 5% FPR, averaged across all the labels.

Model	CW	ElasticNet	PGD	BPDA	SPSA	FGSM
MNIST	95.0%	96.7%	100%	100%	100%	100%
GTSRB	96.3%	100%	100%	100%	93.8%	100%
CIFAR10	100%	97.0%	100%	100%	100%	96.4%
YouTube Face	97.5%	98.8%	100%	100%	96.8%	97.0%

Comparing Trapdoor and Adversarial Perturbation. Our defense is driven by the insight that a trapdoor Δ will trick an adversary into generating an $x + \epsilon$ whose neuron activation vector is similar to $\mathcal{S}_\Delta$, the trapdoor signature. We verify this insight by examining the cosine similarity of $g(x + \epsilon)$ and $\mathcal{S}_\Delta$. We show the results for GTSRB, while the results for other tasks are consistent (see Figure 12 and Figure 13 in the Appendix).

Figure 3(a) plots, for all six attacks against the trapdoored model, the quantile distribution of $cos(g(x+\epsilon), \mathcal{S}_\Delta)$ across x. For reference we also include the result for benign images $cos(g(x), \mathcal{S}_\Delta)$ as the leftmost boxplot. We see that, for all six attacks, the distribution of cosine similarity for adversarial inputs is visibly different from that of benign inputs and thus can be detected by applying a threshold ϕ_t. Furthermore, the distribution of $cos(g(x), \mathcal{S}_\Delta)$ can be used to configure ϕ_t to maximize the adversarial example detection rate at a given false positive rate (FPR).

Figure 3(b) shows the same quantile distribution in the original, trapdoor-free model. As expected, the original model does not produce a clear difference between normal and adversarial inputs. This confirms that the trapdoor can largely affect the shape of adversarial perturbations against the trapdoored model.

Accuracy of Detecting Adversarial Inputs. For all six attacks and all four tasks, Table 1 shows the average adversarial detection success rate when defending a single label. Here we iteratively test our defense on every label in the model, one at a time, and compute the average defense success rate across all the labels[3]. Detection success is $> 93.8\%$ at an FPR of 5% ($> 89\%$ at FPR of 2%). We also show the ROC curves and AUC values in Figure 4 and Figures 7-9 in the Appendix. Across all six attacks and four tasks, detection AUC is $> 98\%$.

Finally, we confirm that a single label trapdoor has negligible impact to model classification on normal inputs.

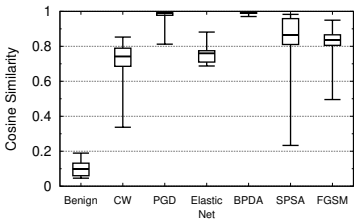
6.3 Defending All Labels

We trained MNIST, GTSRB, CIFAR10, and YouTube Face models with a trapdoor for every output label. Each trapdoor is a randomly selected set of 5 squares (each 3×3 pixels[4]), with $\kappa = 0.1$. The minimum trapdoor injection success rate across the labels is 97% even after injecting 1,283 trapdoors into the YouTube Face model.

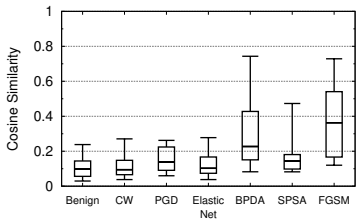
Impact on Normal Classification Accuracy. We first evaluate whether the presence of these trapdoors in the model affects

[3]Due to the large number of labels in the YouTube Face dataset, we randomly sample 100 labels out of 1,283 to defend.

[4]The size of each square is 21 for YouTube Face, which has higher resolution images.



(a) Trapdoored Model



(b) Original Model

Figure 3: Comparison of cosine similarity between normal input/trapdoored inputs and adversarial inputs/trapdoored inputs on both trapdoored and trapdoor-free GTSRB models. Boxes show inter-quartile range and whiskers capture 5^{th} and 95^{th} percentiles.

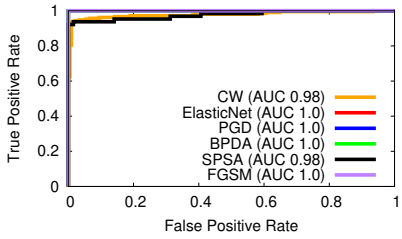


Figure 4: ROC Curve of detection in a GTSRB model when a single label is protected by a trapdoor.

Table 2: Adversarial detection success rate at 5% FPR when defending all labels.

Model	CW	EN	PGD	BPDA	SPSA	FGSM
MNIST	96.8%	98.6%	100%	100%	100%	94.1%
GTSRB	95.6%	96.5%	98.1%	97.6%	97.2%	98.3%
CIFAR10	94.0%	94.0%	100%	99.4%	100%	97.3%
YouTube Face	98.7%	98.2%	100%	97.5%	96.3%	94.8%

the model's normal classification accuracy. We compare the *trapdoored model classification accuracy* to the *original model classification accuracy* on normal inputs in Table 12. The all-label trapdoored model's accuracy on normal inputs drops by at most 1.04% when compared to the original model. This performance drop can potentially be further reduced by optimizing the configuration of trapdoors, which we leave as future work.

Accuracy of Detecting Adversarial Inputs. We run each of the six attacks to find adversarial perturbations against each label of the model and then run our trapdoor-based detection to examine whether an input is adversarial or benign. The adversarial detection success rate is above 94.0% at a FPR of 5% (and 88.3% for FPR of 2%). The detailed results are listed in Table 2.

These results show that, for the all-label defense, adversarial detection accuracy drops slightly compared to the single-label defense. The drop is more visible for YouTube Face, which has significantly more labels (1,283). We believe that as more trapdoors are injected into the model, some of them start to interfere with each other, thus reducing the strength of the shortcuts created in the feature space. This could potentially be ameliorated by carefully placing trapdoors with minimum interference in the feature space. Here, we apply a simple strategy described in Section 4.2 to create separation between trapdoors in the input space. This works well with a few labels (*i.e.* 10, 43). For models with many labels, one can either apply greedy, iterative search to replace "interfering" trapdoor patterns, or develop an accurate metric to capture interference within the injection process. We leave this to future work.

Summary of Observations. For the all-label defense, trapdoor-enabled detection works well across a variety of models and adversarial attack methods. The presence of a large number of trapdoors only slightly degrades normal classification performance. Overall, our defense achieves more than 94% attack detection rate against CW, PGD, ElasticNet, SPSA, FGSM, and more than 97% attack detection rate against BPDA, the strongest known attack.

Table 3: Comparing detection success rate of Feature Squeezing (FS), LID, and Trapdoor when defending all labels.

Model	Detector	FPR	CW	EN	PGD	BPDA	SPSA	FGSM	Avg Succ.
MNIST	FS	5%	99%	100%	94%	96%	94%	98%	97%
	MagNet	5.7%	83%	87%	100%	97%	96%	100%	94%
	LID	5%	89%	86%	96%	86%	98%	95%	92%
	Trapdoor	5%	97%	98%	100%	100%	100%	94%	**98%**
GTSRB	FS	5%	100%	99%	71%	73%	94%	45%	90%
	MagNet	4.7%	90%	89%	100%	100%	92%	100%	95%
	LID	5%	91%	81%	100%	67%	100%	100%	90%
	Trapdoor	5%	96%	97%	98%	98%	97%	98%	**97%**
CIFAR10	FS	5%	100%	100%	69%	66%	97%	33%	78%
	MagNet	7.4%	88%	82%	95%	96%	94%	100%	93%
	LID	5%	90%	88%	95%	79%	96%	92%	90%
	Trapdoor	5%	94%	94%	100%	99%	100%	97%	**97%**
YouTube Face	FS	5%	100%	100%	66%	59%	88%	68%	80%
	MagNet	7.9%	89%	91%	98%	97%	98%	96%	95%
	LID	5%	81%	79%	89%	72%	92%	96%	85%
	Trapdoor	5%	99%	98%	100%	97%	96%	95%	**98%**

6.4 Comparison to Other Detection Methods

Table 3 lists, for all-label defenses, the attack detection AUC for our proposed defense and for three other existing defenses (*i.e.* feature squeezing (FS) [50], MagNet [33], and latent intrinsic dimensionality (LID) [31] described in Section 2.2). For FS, MagNet, and LID, we use the implementations provided by [31, 33, 50]. Again we consider the four tasks and six attack methods as above.

Feature Squeezing (FS). FS can effectively detect gradient-based attacks like CW and ElasticNet, but performs poorly against FGSM, PGD, and BPDA, *i.e.* the detection success rate even drops to 33%. These findings align with existing observations [30, 50].

MagNet. MagNet performs poorly against gradient-based attacks (CW, ElasticNet) but better against FGSM, PGD, and BPDA. This aligns with prior work, which found that adaptive gradient-based attacks can easily defeat MagNet [9].

Latent Intrinsic Dimensionality (LID). LID has $\geq$ 72% detection success rate against all six attacks. In comparison, trapdoor-based detection achieves at least 94% on all six attacks. Like [2], our results also confirm that LID fails to detect high confidence adversarial examples. For example, when we increase the "confidence" parameter of the CW attack from 0 (default) to 50, LID's detection success rate drops to below 2% for all four models. In

comparison, trapdoor-based detection maintains a high detection success rate (97-100%) when confidence varies from 0 to 100. Detection rate reaches 100% when confidence goes above 80. This is because high confidence attacks are less likely to get stuck to local minima and more likely to follow strong "shortcuts" created by the trapdoors.

6.5 Methods for Computing Neuron Signatures

We study how the composition of trapdoor (neuron) signature affects adversarial detection. Recall that, by default, our trapdoor-based detection uses the neuron activation vector right before the softmax layer as the neuron signature of an input. This "signature" is compared to the trapdoor signatures to determine if the input is an adversarial example. In the following, we expand the composition of neuron signatures by varying (1) the internal layer used to extract the neuron signature and (2) the number of neurons used, and examine their impact on attack detection.

First, Figure 10 in Appendix shows the detection success rate when using different layers of the GTSRB model to compute neuron signatures. Past the first two convolutional layers, all later layers lead to detection success greater than 96.20% at 5% FPR. More importantly, choosing any random subset of neurons across these later layers produces an effective activation signature. Specifically, sampling n neurons from any but the first two layers of GTSRB produces an effective trapdoor signature with adversarial detection success rate always above 96%. We find this to be true for a moderate value of $n \sim 900$, much smaller than a single convolutional layer. We confirm that these results also hold for other models, *e.g.* CIFAR10. It is important that small sets of neurons randomly sampled across multiple model layers can build an effective signature. We leverage this flexibility to defend against our final countermeasure (§7.2).

7 ADAPTIVE ATTACKS

Beyond static adversaries, any meaningful defense must withstand countermeasures from adaptive attackers with knowledge of the defense. As discussed in §3.1, we consider two types of adaptive adversaries: *skilled adversaries* who understand the target $\mathcal{F}_\theta$ could have trapdoors without specific knowledge of the details, and *oracle adversaries*, who know all details about embedded trapdoors, including their trapdoor shape, location, and intensity. Since the oracle adversary is the strongest possible adaptive attack, we use its detection rate as the lower bound of our detection effectiveness.

We first present multiple adaptive attacks separated into two broad categories. First, we consider *removal* approaches that attempt to *detect and remove* backdoors from the target model $\mathcal{F}_\theta$, with the eventual intent of generating adversarial examples from the cleaned model, and using them to attack the deployed model $\mathcal{F}_\theta$. Second, we consider *evasion* approaches that do not try to disrupt the trapdoor, and instead focus on finding adversarial examples that cause the desired misclassification while avoiding detection by the trapdoor defense. Our results show that removal approaches fail because the injection of trapdoors largely alters loss functions, and even adversarial examples from the original, trapdoor-free model do not transfer to the trapdoored model.

Finally, we present advanced attacks developed in collaboration with Dr. Nicholas Carlini during the camera ready process. We describe two customized attacks he proposed against trapdoors and show that they effectively break the base version of trapdoors. We also offer preliminary results that show potential mitigation effects via inference-time signature randomization and multiple trapdoors. We leave further exploration of these mechanisms (and more powerful adaptive attacks) to future work.

7.1 Trapdoor Detection and Removal

Backdoor Countermeasures (Skilled Adversary). We start by considering existing work on detecting and removing backdoors from DNNs [27, 28, 37, 48]. A skilled adversary who knows that a target model $\mathcal{F}_\theta$ contains trapdoors may use existing backdoor removal methods to identify and remove them. First, Liu *et al.* proposes to remove backdoors by pruning redundant neurons (*neuron pruning*) [27]. As previous work demonstrates [48], normal model accuracy drops rapidly when pruning redundant neurons. Furthermore, pruning changes the decision boundaries of the pruned model significantly from those of the original model. Hence, adversarial examples that fool the pruned model do not transfer well to the original, since adversarial attacks only transfer between models with similarly decision boundaries [14, 45].

We empirically validated this on a pruned single-label defended MNIST, GTSRB, CIFAR10, and YouTube Face models against the six different attacks. We prune neurons as suggested by [27]. However, we observe that normal accuracy of the model drops rapidly while pruning (> 32.23% drop). Due to the significant discrepancy between the pruned and the original models, adversarial samples crafted on the pruned model do not transfer to the original trapdoored model. Attack success is < 4.67%.

More recently proposed backdoor defenses [28, 37, 48] detect backdoors by finding differences between normal and infected label(s). All of these assume only one or a small number of labels are infected by backdoors, so that they can be identified as anomalies. Authors of *Neural Cleanse* [48] acknowledge that their approach cannot detect backdoors if more than 36% of the labels are infected. Similarly, [37] uses the same technique and has the same limitations. The authors of ABS [28] explicitly state that they do not consider multiple backdoors. We experimentally validate this claim with *Neural Cleanse* against all-label defended versions of MNIST, GTSRB, CIFAR10, YouTube Face. All the trapdoors in our trapdoored models avoided detection.

Black-box/Surrogate Model Attacks (Skilled Adversary). A skilled adversary aware of trapdoors in $\mathcal{F}_\theta$ could use a black-box model stealing attack [34], where they repeatedly query $\mathcal{F}_\theta$ with synthetic, generated inputs, and use the classification results to train a local substitute model. Finally, the adversary generates adversarial examples using the substitute model and used them to attack $\mathcal{F}_\theta$.

Black-box attacks must walk a fine line against trapdoors. To generate adversarial examples that successfully transfer to $\mathcal{F}_\theta$, the attacker must query $\mathcal{F}_\theta$ repeatedly with inputs close to the classification boundary. Yet doing so means that black-box attacks could also import the trapdoors of $\mathcal{F}_\theta$ into the substitute model.

We test the effectiveness of black box attacks by defending single label GTSRB models as described in Section 6.2. We construct the substitute model following [34] and use it to generate adversarial attack images to attack our original model $\mathcal{F}_\theta$. In our tests, we

Table 4: Targeted transferability of Adversarial Examples from a model restored by unlearning, to its trapdoored counterpart.

Model	CW	EN	PGD	BPDA	SPSA	FGSM
GTSRB	1.5%	2.6%	2.0%	1.0%	0.0%	4.7%
CIFAR10	4.4%	4.4%	5.6%	0.0%	6.7%	0.0%
Youtube Face	0.0%	0.0%	4.1%	3.3%	0.0%	0.0%

Table 5: Targeted transferability of Adversarial Examples from a model trained on clean data to its trapdoored counterpart.

Model	CW	EN	PGD	BPDA	SPSA	FGSM
GTSRB	0.0%	0.0%	2.2%	3.0%	1.0%	0.4%
CIFAR10	0.0%	0.0%	1.7%	0.7%	2.8%	1.2%
Youtube Face	0.0%	0.0%	2.1%	1.7%	0.0%	0.0%

consistently observe that the substitute model does indeed inherit the trapdoors from $\mathcal{F}_\theta$. A trapdoored model can reliably detect adversarial examples generated from black-box substitute models with $>$ 95% success at 5% false positive rate, for all six attacks (FGSM, PGD, CW, EN, BPDA, SPSA).

If somehow an attacker obtained access to the full training dataset used by the model and used it to build a surrogate model, they could reproduce the original clean model. We consider this possibility later in this subsection.

Unlearning the Trapdoor (Oracle Adversary). The goal of this countermeasure is to completely remove trapdoors from the target model $\mathcal{F}_\theta$ so that attackers can use it to generate adversarial samples to attack $\mathcal{F}_\theta$. Prior work has shown that adversarial attacks can transfer between models trained on similar data [14, 45]. This implies that attacks may transfer between cleaned and trapdoored versions of the target model.

For this we consider an *oracle attacker* who knows everything about a model's embedded trapdoors, including its exact shape and intensity. With such knowledge, oracle adversaries seek to construct a trapdoor-free model by unlearning the trapdoors.

However, we find that such a transfer attack (between $\mathcal{F}_\theta$ and a version of it with the trapdoor unlearned $\mathcal{F}_\theta{}^{unlearn}$) fails. We validate this experimentally using a single-label defended model. The high level results are summarized in Table 4. We create a new version of each trapdoored model using backdoor unlearning techniques [5, 48], which reduce the trapdoor injection success rate from 99% to negligible rates (around 2%). Unsurprisingly, the trapdoor defense is unable to detect adversarial samples constructed on the cleaned model $\mathcal{F}_\theta{}^{unlearn}$, with only 7.42% detection success rate at 5% FPR for GTSRB. However, these undetected adversarial samples do not transfer to the trapdoored model $\mathcal{F}_\theta$. For all six attacks and all four models, the attack success rate on $\mathcal{F}_\theta$ ranges from 0% to 6.7%. We hypothesize that this might be because a trapdoored model $\mathcal{F}_\theta$ must learn unique *trapdoor distributions* that $\mathcal{F}_\theta{}^{unlearn}$ does not know. This distributional shift causes significant differences that are enough to prevent adversarial examples from transferring between models.

Oracle Access to the Original Clean Model. Unlearning is unlikely to precisely recover the original clean model (before the trapdoor). Finally, we consider the strongest removal attack possible: an oracle attacker that has somehow obtained access to (or perfectly reproduced) the original clean model. We evaluate the impact of using the original clean model to generate adversarial attacks on $\mathcal{F}_\theta$.

We are surprised to learn that adding the trapdoor has introduced significant changes in the original clean model, and has thus destroyed the transferability of adversarial attacks between them. In Table 5, we show the transferability from clean models to their trapdoored counterparts. For all 6 attacks and all models, transferability is always never higher than 3%. This definitive result states that no matter how successful an attacker is at removing or unlearning the trapdoor, or if they otherwise rebuild the original model, their efforts will fail because adversarial examples from these models do not work on the trapdoored model $\mathcal{F}_\theta$ that is the actual attack target.

7.2 Advanced Adaptive Attacks (Carlini)

In this section, we present results on two advanced attacks developed in collaboration with Dr. Nicholas Carlini, both crafted to detect and leverage weaknesses in the design of trapdoors. Nicholas approached us after the paper was accepted and offered to test the robustness of trapdoors by developing more advanced adaptive attacks. Both attacks are significantly more successful in weakening trapdoor defenses. Here, we describe both attacks, their key approaches and their results on different types of trapdoor defenses.

We note that a prior version of the paper included results on two other adaptive attacks: a low learning rate attack that more carefully scans the loss landscape for adversarial examples, and a lower-bound perturbation attack that tries to avoid trapdoors by imposing a lower bound on the size of the perturbation. Our results show both attacks are largely ineffective against trapdoors. Due to space constraints, we focus on two stronger Carlini attacks here, and refer readers to [40] for detailed results on low learning rate and lower-bound perturbation attacks.

Generalities. Nicholas' two attacks share two general principles. First, they use different techniques to map out the boundaries of trapdoors that exist in a protected model, *i.e.* their detection signatures, and then devise ways to compute adversarial perturbations that avoid them. Second, they leverage significant compute power, well beyond normal experimental levels, *e.g.* running 10K optimization iterations instead of terminating on convergence. We consider these quite reasonable for an attacker and do not consider computational overhead a mitigating factor.

Instead, we evaluaese attacks against variants of trapdoors previously discussed: randomized neuron signatures (§6.5) and multiple trapdoors per label (§4.1). First, randomized neuron signatures can effectively make the signature dynamic at run time. Since trapdoor avoidance is likely a greedy operation, the inclusion or exclusion of a small number of neurons can significantly alter the result. In practice, the defender can quickly switch between different neuron subsets in unpredictable sequences, making attacker optimizations difficult. Second, multiple trapdoors cover more of the feature space, making trapdoor avoidance more difficult. In addition, we hypothesize that additional trapdoors will remove some natural adversarial examples, much like a randomized smoothing

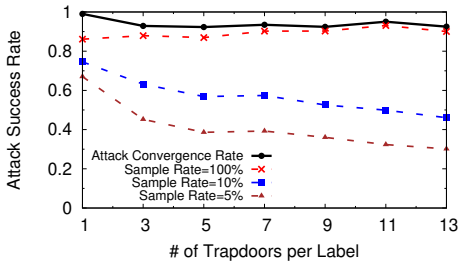


Figure 5: Oracle Signature Attack success against random neuron sampling and multiple trapdoors.

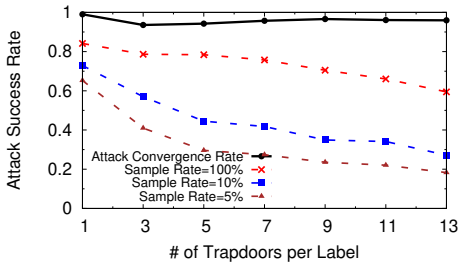


Figure 6: Trapdoor Vault Attack success against random neuron sampling and multiple trapdoors.

function. When we evaluate using multiple trapdoors, we assume the attacker knows exactly the number of trapdoors that have been inserted into the model per label. Note that we generated 5 models for each trapdoor configuration to eliminate variance in results due to randomness in trapdoor placement and then performed 10 attacks on each model to generate each single data point.

Oracle Signature Attack. Nicholas' first attack uses our oracle adversary assumptions, where the attacker knows the precise values of the trapdoor. Combined with whitebox access to the model, the attacker is thus able to generate neuron values matching input with the trapdoor for any layer in the model. The attacker then jointly optimizes for maximum cosine distance from the known trapdoor signature while searching for adversarial examples.

We plot in Figure 5 the results of the oracle signature attack on MNIST with both neuron randomization and multiple trapdoors per label for all label defense. The top (black) line shows the attack's ability to find successful adversarial examples. The red line marked as 100% sample rate represents the attack success after detection by trapdoor defense without neuron randomization. Without randomized neuron sampling, the signature attack succeeds nearly 90% of the time, effectively breaking the trapdoor defense. The other two lines show the attack success rate using neuron randomization. We do see that neuron sampling helps to mitigate the attack, reducing attack success to 40% after detection, by using only 5% of neuron values to build the detection signature. Recall that the defender can update the random neuron subset in real time with near zero computational cost. We note that additional trapdoors have only a slight effect on detection, because the attacker always knows the number of trapdoors present.

Trapdoor Vault Attack. Nicholas' second attack, which we call the *vault attack*, does not require knowledge of the trapdoor signature. Instead, it estimates the trapdoor signature from the results of computing adversarial examples on multiple input images. We extrapolate this further to target multiple trapdoors by using a histogram/clustering approach to approximate neuron signatures for each of the N trapdoors. The attacker then computes joint optimization that maximizes distance to known trapdoor signatures while searching for adversarial examples. Again we assume attackers know the exact number of trapdoors present in the model.

We plot in Figure 6 the results of the vault attack on MNIST with both neuron randomization and multiple trapdoors. Again, we see only small benefits from having multiple trapdoors in the model. However, in this setting the trapdoor defense does detect more attacks because of errors in the signature approximation (which can likely be improved with effort). We do note that when combining randomized neuron sampling (at 5%) with multiple trapdoors, we can detect significantly more attacks, dropping attack success to below 40%.

Discussion and Next Steps. Time constraints greatly limited the amount of exploration possible in both mitigation mechanisms and further adaptive attacks. Under base conditions (single trapdoor with 100% neuron signature sampling), both attacks effectively break the trapdoor defense. While our preliminary results show some promise of mitigation, clearly much more work is needed to explore additional defenses (and more powerful adaptive attacks).

These attacks are dramatically more effective than other countermeasures because they were custom-tailored to target trapdoors. We consider their efficacy as validation that defense papers should work harder to include more rigorous, targeted adaptive attacks.

8 CONCLUSION AND FUTURE WORK

In this paper, we propose using honeypots to defend DNNs against adversarial examples. Unlike traditional defenses, our proposed method trains trapdoors into normal models to introduce controlled vulnerabilities (traps) into the model. Trapdoors can defend all labels or particular labels of interest. Across multiple application domains, our trapdoor-based defense has high detection success against adversarial examples generated by a suite of state-of-the-art adversarial attacks, including CW, ElasticNet, PGD, BPDA, FGSM, and SPSA, with negligible impact on normal input classification.

In addition to analytical proofs of the impact of trapdoors on adversarial attacks, we evaluate and confirm trapdoors' robustness against multiple strong adaptive attacks, including black-box attacks and unlearning attacks. Our results on Carlini's oracle and vault attacks show that trapdoors do have significant vulnerabilities. While randomized neuron signatures help mitigation, it is clear that further effort is necessary to study both stronger attacks and mitigation strategies on honeypot-based defenses.

ACKNOWLEDGMENTS

We are thankful for significant time and effort contributed by Nicholas Carlini in helping us develop stronger adaptive attacks on trapdoors. We have learned much in the process. We also thank our shepherd Ting Wang and anonymous reviewers for their constructive feedback. This work is supported in part by NSF grants CNS-1949650, CNS-1923778, CNS-1705042, and by the DARPA GARD program. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of any funding agencies.

REFERENCES

[1] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. 2018. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *Proc. of USENIX Security*.

[2] Anish Athalye, Nicholas Carlini, and David Wagner. 2018. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *Proc. of ICML*.

[3] J. Buckman, A. Roy, C. Raffel, and I. Goodfellow. 2018. Thermometer encoding: One hot way to resist adversarial examples. In *Proc. of ICLR*.

[4] Qiong Cao, Li Shen, Weidi Xie, Omkar M Parkhi, and Andrew Zisserman. 2018. Vggface2: A dataset for recognising faces across pose and age. In *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*. IEEE, 67–74.

[5] Yinzhi Cao, Alexander Fangxiao Yu, Andrew Aday, Eric Stahl, Jon Merwine, and Junfeng Yang. 2018. Efficient Repair of Polluted Machine Learning Systems via Causal Unlearning. In *Proc. of ASIACCS*.

[6] Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian Goodfellow, Aleksander Madry, and Alexey Kurakin. 2019. On Evaluating Adversarial Robustness. *arXiv preprint arXiv:1902.06705* (2019).

[7] Nicholas Carlini and David Wagner. 2016. Defensive distillation is not robust to adversarial examples. *arXiv preprint arXiv:1607.04311* (2016).

[8] Nicholas Carlini and David Wagner. 2017. Adversarial examples are not easily detected: Bypassing ten detection methods. *Proc. of AISec* (2017).

[9] Nicholas Carlini and David Wagner. 2017. Magnet and efficient defenses against adversarial attacks are not robust to adversarial examples. *arXiv preprint arXiv:1711.08478* (2017).

[10] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *Proc. of IEEE S&P*.

[11] Antonin Chambolle. 2004. An algorithm for total variation minimization and applications. *Journal of Mathematical Imaging and Vision* 20, 1 (2004), 89–97.

[12] Pin-Yu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. 2018. EAD: elastic-net attacks to deep neural networks via adversarial examples. In *Proc. of AAAI*.

[13] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. 2017. Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning. *arXiv preprint arXiv:1712.05526* (2017).

[14] Ambra Demontis, Marco Melis, Maura Pintor, Matthew Jagielski, Battista Biggio, Alina Oprea, Cristina Nita-Rotaru, and Fabio Roli. 2019. Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks. In *Proc. of USENIX Security*. 321–338.

[15] G. S. Dhillon, K. Azizzadenesheli, J. D. Bernstein, J. Kossaifi, A. Khanna, Z. C. Lipton, and A. Anandkumar. 2018. Stochastic activation pruning for robust adversarial defense. In *Proc. of ICLR*.

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

[17] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. 2017. Badnets: Identifying vulnerabilities in the machine learning model supply chain. In *Proc. of Machine Learning and Computer Security Workshop*.

[18] C. Guo, M. Rana, M. Cisse, and L. van der Maaten. 2018. Countering adversarial images using input transformations. In *Proc. of ICLR*.

[19] Yandong Guo, Lei Zhang, Yuxiao Hu, Xiaodong He, and Jianfeng Gao. 2016. Ms-celeb-1m: A dataset and benchmark for large-scale face recognition. In *European Conference on Computer Vision*. Springer, 87–102.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proc. of CVPR*.

[21] Warren He, James Wei, Xinyun Chen, Nicholas Carlini, and Dawn Song. 2017. Adversarial example defenses: Ensembles of weak defenses are not strong. In *Proc. of WOOT*.

[22] J. Zico Kolter and Eric Wong. 2017. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *Proc. of NeurIPS*.

[23] Alex Krizhevsky and Geoffrey Hinton. 2009. *Learning multiple layers of features from tiny images*. Technical Report.

[24] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2016. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533* (2016).

[25] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2017. Adversarial machine learning at scale. In *Proc. of ICLR*.

[26] Yann LeCun, LD Jackel, Léon Bottou, Corinna Cortes, John S Denker, Harris Drucker, Isabelle Guyon, UA Muller, Eduard Sackinger, Patrice Simard, et al. 1995. Learning algorithms for classification: A comparison on handwritten digit recognition. *Neural Networks* 261 (1995), 276.

[27] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. 2018. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 273–294.

[28] Yingqi Liu, Wen-Chuan Lee, Guanhong Tao, Shiqing Ma, Yousra Aafer, and Xiangyu Zhang. 2019. ABS: Scanning neural networks for back-doors by artificial brain stimulation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1265–1282.

[29] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. 2018. Trojaning Attack on Neural Networks. In *Proc. of NDSS*.

[30] Shiqing Ma, Yingqi Liu, Guanhong Tao, Wen-Chuan Lee, and Xiangyu Zhang. 2019. NIC: Detecting Adversarial Samples with Neural Network Invariant Checking. In *Proc. of NDSS*.

[31] Xingjun Ma, Bo Li, Yisen Wang, Sarah M Erfani, Sudanthi Wijewickrema, Grant Schoenebeck, Dawn Song, Michael E Houle, and James Bailey. 2018. Characterizing adversarial subspaces using local intrinsic dimensionality. In *Proc. of ICLR*.

[32] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards deep learning models resistant to adversarial attacks. In *Proc. of ICLR*.

[33] Dongyu Meng and Hao Chen. 2017. Magnet: a two-pronged defense against adversarial examples. In *Proc. of CCS*.

[34] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. 2017. Practical black-box attacks against machine learning. In *Proc. of AsiaCCS*.

[35] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2016. Distillation as a defense to adversarial perturbations against deep neural networks. In *Proc. of IEEE S&P*.

[36] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, et al. 2015. Deep face recognition.. In *bmvc*, Vol. 1. 6.

[37] Ximing Qiao, Yukun Yang, and Hai Li. 2019. Defending Neural Backdoors via Generative Distribution Modeling. *arXiv preprint arXiv:1910.04749* (2019).

[38] P. Samangouei, M. Kabkab, and R. Chellappa. 2018. Defensegan: Protecting classifiers against adversarial attacks using generative models. In *Proc. of ICLR*.

[39] Ali Shafahi, W Ronny Huang, Christoph Studer, Soheil Feizi, and Tom Goldstein. 2019. Are adversarial examples inevitable?. In *Proc. of ICLR*.

[40] Shawn Shan, Emily Wenger, Bolun Wang, Bo Li, Haitao Zheng, and Ben Y. Zhao. 2020. Gotta Catch 'Em All: Using Honeypots to Catch Adversarial Attacks on Neural Networks. *arXiv preprint: 1904.08554* (2020).

[41] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K Reiter. 2016. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proc. of CCS*.

[42] Y. Song, T. Kim, S. Nowozin, S. Ermon, and N. Kushman. 2018. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. In *Proc. of ICLR*.

[43] James C Spall et al. 1992. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Trans. Automat. Control* 37, 3 (1992), 332–341.

[44] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. 2012. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks* (2012).

[45] Octavian Suciu, Radu Mărginean, Yiğitcan Kaya, Hal Daumé III, and Tudor Dumitraş. 2018. When Does Machine Learning FAIL? Generalized Transferability for Evasion and Poisoning Attacks. In *Proc. of USENIX Security*.

[46] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *Proc. of ICLR*.

[47] Jonathan Uesato, Brendan O'Donoghue, Aaron van den Oord, and Pushmeet Kohli. 2018. Adversarial risk and the dangers of evaluating against weak attacks. *arXiv preprint arXiv:1802.05666* (2018).

[48] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y. Zhao. 2019. Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks. In *Proc. of IEEE S&P*.

[49] C. Xie, J. Wang, Z. Zhang, Z. Ren, and A. Yuille. 2018. Mitigating adversarial effects through randomization. In *Proc. of ICLR*.

[50] Weilin Xu, David Evans, and Yanjun Qi. 2018. Feature squeezing: Detecting adversarial examples in deep neural networks. In *Proc. of NDSS*.

[51] YouTube [n.d.]. https://www.cs.tau.ac.il/~wolf/ytfaces/. YouTube Faces DB.

[52] Valentina Zantedeschi, Maria-Irina Nicolae, and Ambrish Rawat. 2017. Efficient defenses against adversarial attacks. In *Proc. of AISec*.

[53] Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph Stoecklin, Heqing Huang, and Ian Molloy. 2018. Protecting intellectual property of deep neural networks with watermarking. In *Proc. of AsiaCCS*.

[54] Stephan Zheng, Yang Song, Thomas Leung, and Ian Goodfellow. 2016. Improving the robustness of deep neural networks via stability training. In *Proc. of CVPR*.

Table 6: Model Architecture for MNIST. FC stands for fully-connected layer.

Layer Type	# of Channels	Filter Size	Stride	Activation
Conv	16	5×5	1	ReLU
MaxPool	16	2×2	2	-
Conv	32	5×5	1	ReLU
MaxPool	32	2×2	2	-
FC	512	-	-	ReLU
FC	10	-	-	Softmax

Table 7: Model Architecture of GTSRB.

Layer Type	# of Channels	Filter Size	Stride	Activation
Conv	32	3×3	1	ReLU
Conv	32	3×3	1	ReLU
MaxPool	32	2×2	2	-
Conv	64	3×3	1	ReLU
Conv	64	3×3	1	ReLU
MaxPool	64	2×2	2	-
Conv	128	3×3	1	ReLU
Conv	128	3×3	1	ReLU
MaxPool	128	2×2	2	-
FC	512	-	-	ReLU
FC	43	-	-	Softmax

APPENDIX

8.1 Proof of Theorem 1 & 2

Proof of Theorem 1

Proof. This theorem assumes that after injecting the trapdoor Δ into the model, we have

$$\forall x \in \mathcal{X}, \quad \Pr\left(\mathcal{F}_\theta(x+\Delta) = y_t \neq \mathcal{F}_\theta(x)\right) \geq 1-\mu. \tag{6}$$

When an attacker applies gradient-based optimization to find adversarial perturbations for an input x targeting y_t, the above equation (6) implies that the partial gradient from x towards $x+\Delta$ becomes the major gradient to achieve the target y_t. Note that $\mathcal{F}_\theta(x)$ is the composition of non-linear feature representation $g(x)$ and a linear loss function (e.g. logistic regression): $\mathcal{F}_\theta(x) = g(x) \circ L$ where L represents the linear function. Therefore, the gradient of $\mathcal{F}_\theta(x)$ can be calculated via $g(x)$:

$$\frac{\partial ln\mathcal{F}_\theta(x)}{\partial x} = \frac{\partial ln[g(x) \circ L]}{\partial x} = c\frac{\partial lng(x) \circ L}{\partial x} \tag{7}$$

Here c is the constant within the linear function L. To avoid ambiguity, we will focus on the derivative on $g(x)$ in the rest of the proof.

Given (7), we can interpret (6) in terms of the major gradient:

$$P_{x\in\mathcal{X}}[\frac{\partial[lng(x) - lng(x+\Delta)]}{\partial x} \geq \eta] \geq 1-\mu, \tag{8}$$

where η represents, for the given x, the gradient value required to reach y_t as the classification result.

Next, since $\forall x \in \mathcal{X}$, $cos(g(A(x)), g(x+\Delta)) \geq \sigma$, and $\sigma \to 1$, without loss of generality we have $g(A(x)) = g(x+\Delta) + \gamma$ where $|\gamma| << |g(x+\Delta)|$. Here we rewrite the adversarial input $A(x)$ as $A(x) = x + \epsilon$. Using this condition, we can prove that the following two conditions are true. First, because the value of γ does not depend on x, we have

$$\frac{\partial(g(x+\Delta)+\gamma)}{\partial x} = \frac{\partial g(x+\Delta)}{\partial x}. \tag{9}$$

Furthermore, because $|\gamma| << |g(x+\Delta|)$, we have

$$\frac{1}{g(x+\Delta)+\gamma} \approx \frac{1}{g(x+\Delta)}. \tag{10}$$

Leveraging eq. (8)-(10), we have

$$\begin{aligned}
&P_{x\in\mathcal{X}}[\frac{\partial[lng(x) - lng(x+\epsilon)]}{\partial x} \geq \eta] \\
=&P_{x\in\mathcal{X}}[\frac{1}{g(x)}\frac{\partial g(x)}{\partial x} - \frac{1}{g(x+\epsilon)}\frac{\partial g(x+\epsilon)}{\partial x} \geq \eta] \\
=&P_{x\in\mathcal{X}}[\frac{1}{g(x)}\frac{\partial g(x)}{\partial x} - \frac{1}{g(x+\Delta)+\gamma}\frac{\partial(g(x+\Delta)+\gamma)}{\partial x} \geq \eta] \\
\approx&P_{x\in\mathcal{X}}[\frac{1}{g(x)}\frac{\partial g(x)}{\partial x} - \frac{1}{g(x+\Delta)}\frac{\partial(g(x+\Delta))}{\partial x} \geq \eta] \\
=&P_{x\in\mathcal{X}}[\frac{\partial[lng(x) - lng(x+\Delta)]}{\partial x} \geq \eta] \\
\geq&1-\mu.
\end{aligned}$$

□

Proof of Theorem 2

Proof. This theorem assumes that, after injecting the trapdoor Δ, we have

$$P_{x\in\mathcal{X}_{trap}}[\frac{\partial[lng(x) - lng(x+\Delta)]}{\partial x} \geq \eta] \geq 1-\mu \tag{11}$$

Following the same proof procedure in Theorem 1, we have

$$P_{x\in\mathcal{X}_{trap}}[\frac{\partial[lng(x) - lng(x+\epsilon)]}{\partial x} \geq \eta] \geq 1-\mu \tag{12}$$

Since $\mathcal{X}_{trap}$ and $\mathcal{X}_{attack}$ are ρ-*covert*, by definition (see eq. (5)) we have that for any event $C \subset \Omega$, the largest possible difference between the following probabilities $P_{x\in\mathcal{X}_{attack}}[C]$ and $P_{x\in\mathcal{X}_{trap}}[C]$ is bounded by ρ.

Next let C represent the event: $(\frac{\partial[lng(x)-lng(x+\epsilon)]}{\partial x} \geq \eta)$. We have, for $x \in \mathcal{X}_{attack}$,

$$\begin{aligned}
&P_{x\in\mathcal{X}_{attack}}[\frac{\partial[lng(x) - lng(x+\epsilon)]}{\partial x} \geq \eta] \\
\geq&P_{x\in\mathcal{X}_{trap}}[\frac{\partial[lng(x) - lng(x+\epsilon)]}{\partial x} \geq \eta] - \rho \\
\geq&1-(\mu+\rho).
\end{aligned}$$

□

8.2 Experiment Configuration

Evaluation Dataset. We discuss in details of training datasets we used for the evaluation.

- *Hand-written Digit Recognition* (MNIST) – This task seeks to recognize 10 handwritten digits (0-9) in black and white images [26]. The dataset consists of 60,000 training images and 10,000 test images. The DNN model is a standard 4-layer convolutional neural network (see Table 6).

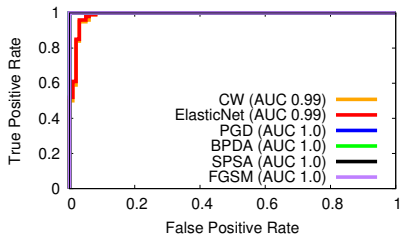


Figure 7: ROC Curve of detection on MNIST with single-label defense.

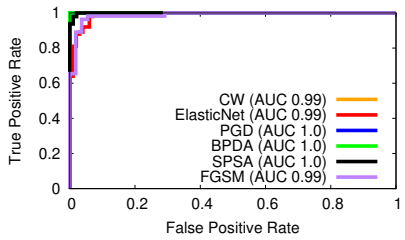


Figure 8: ROC Curve of detection on CIFAR10 with single-label defense.

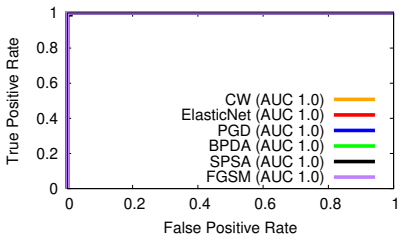


Figure 9: ROC Curve of detection on YouTube Face with single-label defense.

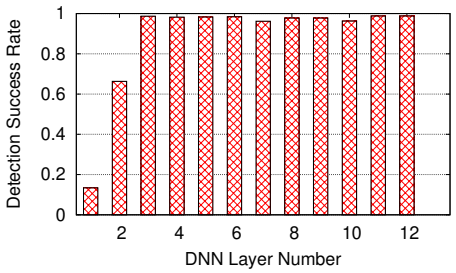


Figure 10: Detection success rate of CW attack at 5% FPR when using different layers for detection in a GTSRB model.

Table 8: ResNet20 Model Architecture for CIFAR10.

Layer Name (type)	# of Channels	Activation	Connected to
conv_1 (Conv)	16	ReLU	-
conv_2 (Conv)	16	ReLU	conv_1
conv_3 (Conv)	16	ReLU	pool_2
conv_4 (Conv)	16	ReLU	conv_3
conv_5 (Conv)	16	ReLU	conv_4
conv_6 (Conv)	16	ReLU	conv_5
conv_7 (Conv)	16	ReLU	conv_6
conv_8 (Conv)	32	ReLU	conv_7
conv_9 (Conv)	32	ReLU	conv_8
conv_10 (Conv)	32	ReLU	conv_9
conv_11 (Conv)	32	ReLU	conv_10
conv_12 (Conv)	32	ReLU	conv_11
conv_13 (Conv)	32	ReLU	conv_12
conv_14 (Conv)	32	ReLU	conv_13
conv_15 (Conv)	64	ReLU	conv_14
conv_16 (Conv)	64	ReLU	conv_15
conv_17 (Conv)	64	ReLU	conv_16
conv_18 (Conv)	64	ReLU	conv_17
conv_19 (Conv)	64	ReLU	conv_18
conv_20 (Conv)	64	ReLU	conv_19
conv_21 (Conv)	64	ReLU	conv_20
pool_1 (AvgPool)	-	-	conv_21
dropout_1 (Dropout)	-	-	pool_1
fc_ (FC)	-	Softmax	dropout_1

- *Traffic Sign Recognition* (GTSRB) – Here the goal is to recognize 43 different traffic signs, emulating an application for self-driving cars. We use the German Traffic Sign Benchmark dataset (GTSRB), which contains 35.3K colored training images and 12.6K testing images [44]. The model consists of 6 convolution layers and 2 dense layers (see Table 7). This task is 1) commonly used as an adversarial defense evaluation benchmark and 2) represents a real-world setting relevant to our defense.
- *Image Recognition* (CIFAR10) – The task is to recognize 10 different objects. The dataset contains 50K colored training images and 10K testing images [23]. The model is an Residual Neural Network (RNN) with 20 residual blocks and 1 dense layer [20] (Table 8). We include this task because of its prevalence in general image classification and adversarial defense literature.
- *Face Recognition* (YouTube Face) – This task is to recognize faces of 1,283 different people drawn from the YouTube videos [51]. We build the dataset from [51] to include 1,283 labels, 375.6K training images, and 64.2K testing images [13]. We use a large ResNet-50 architecture architecture [20] with over 25 million parameters. We include this task because it simulates a more complex facial recognition-based security screening scenario. Defending against adversarial attack in this setting is important. Furthermore, the large set of labels in this task allows us to explore the scalability of our trapdoor-enabled detection.

Model Architecture. We now present the architecture of DNN models used in our work.

- **MNIST** (Table 6) is a convolutional neural network (CNN) consisting of two pairs of convolutional layers connected by max pooling layers, followed by two fully connected layers.
- **GTSRB** (Table 7) is a CNN consisting of three pairs of convolutional layers connected by max pooling layers, followed by two fully connected layers.
- **CIFAR10** (Table 8) is also a CNN but includes 21 sequential convolutional layers, followed by pooling, dropout, and fully connected layers.
- **YouTube Face** is the ResNet-50 model trained on the YouTube Face dataset. It has 50 residual blocks with over 25 millions parameters.

Detailed information on attack configuration. We evaluate the trapdoor-enabled detection using six adversarial attacks: CW, ElasticNet, PGD, BPDA, SPSA, and FGSM (which we have described in Section 2.1). Details about the attack configuration are listed in Table 10.

Sample Trapdoor Patterns. Figure 11 shows sample images that contain a single-label defense trapdoor (a single 6×6 square) and that contain an all-label defense trapdoor (five 3×3 squares). The mask ratio of the trapdoors used in our experiments is fixed to $\kappa = 0.1$.

Table 9: Detailed information on datasets and defense configurations for each trapdoored model when protecting all labels.

Model	# of Labels	Training Set Size	Testing Set Size	Injection Ratio	Mask Ratio	Training Configuration
MNIST	10	50,000	10,000	0.5	0.1	epochs=5, batch=32, optimizer=Adam, lr=0.001
GTSRB	43	35,288	12,630	0.5	0.1	epochs=30, batch=32, optimizer=Adam, lr=0.001
CIFAR10	10	50,000	10,000	0.5	0.1	epochs=60, batch=32, optimizer=Adam, lr=0.001
YouTube Face	1,283	375,645	64,150	0.5	0.2	epochs=30, batch=32, optimizer=Adam, lr=0.001

Table 10: Detailed information on attack configurations. For MNIST experiments, we divid the eps value by 255.

Attack Method	Attack Configuration
CW	binary step size = 9, max iterations = 1000, learning rate = 0.05, abort early = True
PGD	max eps = 8, # of iteration = 100, eps of each iteration = 0.1
ElasticNet	binary step size = 20, max iterations = 1000, learning rate = 0.5, abort early = True
BPDA	max eps = 8, # of iteration = 100, eps of each iteration = 0.1
SPSA	eps = 8, # of iteration = 500, learning rate = 0.1
FGSM	eps = 8

(a) Single Label Defense Trapdoor (b) All Label Defense Trapdoor

Figure 11: Sample trapdoor examples used in our defense. While the actual trapdoors we used all have a mask ratio of $\kappa = 0.1$, here we artifically increase κ from 0.1 to 1.0 in order to highlight the trapdoors from the rest of the image content.

Table 11: Dataset, complexity, model architecture for each task.

Task	Dataset	# of Labels	Input Size	Training Images	Model Architecture
Digit Recognition	MNIST	10	$28 \times 28 \times 1$	60,000	2 Conv, 2 Dense [6]
Traffic Sign Recognition	GTSRB	43	$32 \times 32 \times 3$	35,288	6 Conv, 2 Dense [7]
Image Recognition	CIFAR10	10	$32 \times 32 \times 3$	50,000	20 Resid, 1 Dense [8]
Facial Recognition	YouTube Face	1,283	$224 \times 224 \times 3$	375,645	ResNet-50 [20]

Table 12: Trapdoored model and original model classification accuracy when injecting trapdoors for all labels.

Model	Original Model Classification Accuracy	Trapdoored Model (All Labels) Classification Accuracy
MNIST	99.2%	98.6%
GTSRB	97.3%	96.3%
CIFAR10	87.3%	86.9%
YouTube Face	99.4%	98.8%

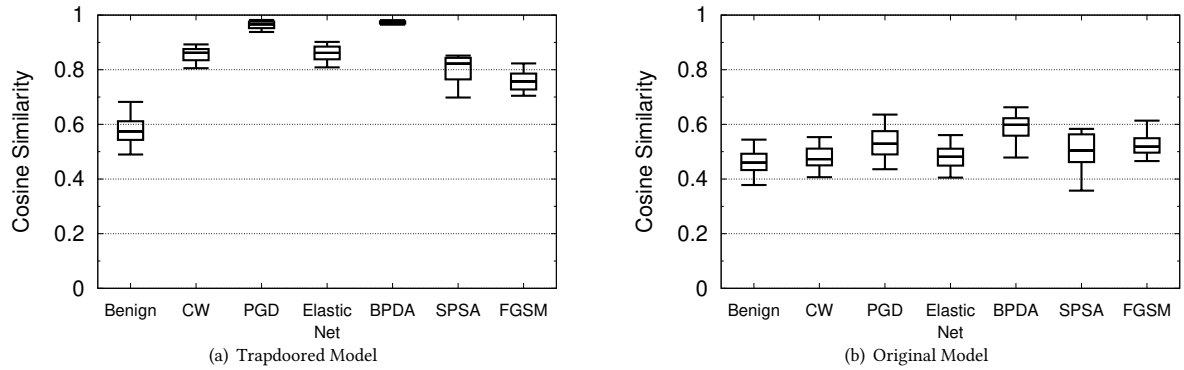
Datasets and Defense Configuration. Tablel 9 lists the specific datasets and training process used to inject trapdoors into the four DNN models.

Selecting Trapdoor Injection Ratio. As mentioned earlier, our analysis shows that the size and diversity of the training data used to inject a trapdoor could affect its effectivess of trapping attackers. To explore this factor, we define *trapdoor injection ratio* as the ratio between the trapdoored images and the clean images in the training dataset. Intuitively, a higher injection ratio should allow the model to learn the trapdoor better but could potentially degrade normal classification accuracy.

We defend the model with different trapdoor injection ratios and examine the detection success rate. We see that only when the injection ratio is very small (*e.g.* < 0.03 for GTSRB), the model fails to learn the trapdoor and therefore detection fails. Otherwise the trapdoor is highly effective in terms of detecting adversarial examples. Thus when building the trapdoored models, we use an injection ratio of 0.1 for MNIST, GTSRB, CIFAR1010, and 0.01 for YouTube Face (see Table 10).

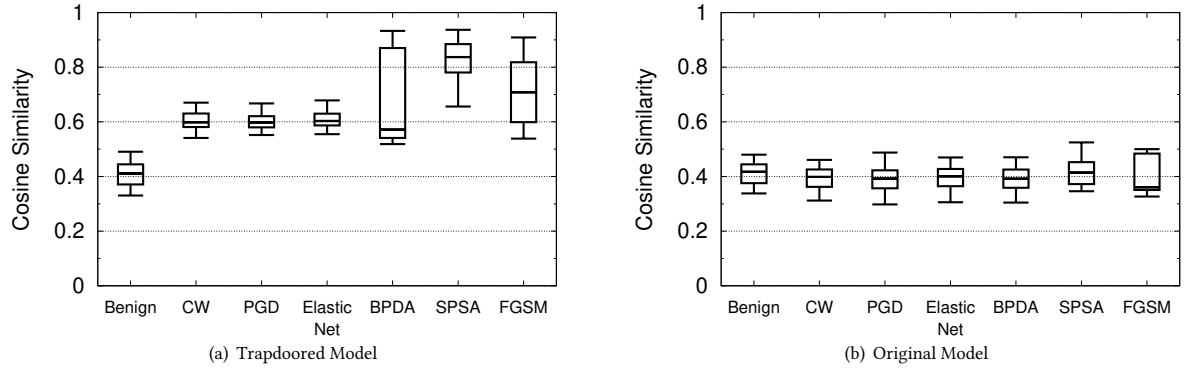
8.3 Additional Results on Comparing Trapdoor and Adversarial Perturbation

Figure 12 and Figure 13 show that the neuron signatures of adversarial inputs have high cosine similarity to the neuron signatures of trapdoors in a trapdoored CIFAR10 and YouTube Face models (left figures), and the trapdoor-free models (right figures).



(a) Trapdoored Model (b) Original Model

Figure 12: Comparison of cosine similarity of normal images and adversarial images to trapdoored inputs in a trapdoored CIFAR10 model and in an original (trapdoor-free) CIFAR10 model. The boxes show the inter-quartile range, and the whiskers denote the 5^{th} and 95^{th} percentiles.



(a) Trapdoored Model (b) Original Model

Figure 13: Comparison of cosine similarity of normal images and adversarial images to trapdoored inputs in a trapdoored YouTube Face model and in an original (trapdoor-free) YouTube Face model. The boxes show the inter-quartile range, and the whiskers denote the 5^{th} and 95^{th} percentiles.