STRIP: A Defence Against Trojan Attacks on Deep Neural Networks

Yansong Gao
yansong.gao@njust.edu.cn
School of Computer Science and Engineering, Nanjing University of Science and Technology, China and Data61, CSIRO, Sydney, Australia.

Change Xu
chang.xu@data61.csiro.au
Data61, CSIRO, Sydney, Australia.

Derui Wang
deruiwang@swin.edu.au
Swinburne University of Technology and Data61, CSIRO, Australia.

Shiping Chen
shiping,chen@data61.csiro.au
Data61, CSIRO, Sydney, Australia

Damith C. Ranasinghe
damith.ranasinghe@adelaide.edu.au
Auto-ID Labs, School of Computer Science, The University of Adelaide, SA 5005, Australia.

Surya Nepal
surya,nepal@data61.csiro.au
Data61, CSIRO, Sydney, Australia

ABSTRACT

A recent trojan attack on deep neural network (DNN) models is one insidious variant of data poisoning attacks. Trojan attacks exploit an effective *backdoor* created in a DNN model by leveraging the difficulty in interpretability of the learned model to misclassify *any inputs* signed with the attacker's chosen trojan trigger. Since the trojan trigger is a secret guarded and exploited by the attacker, detecting such *trojan inputs* is a challenge, especially at run-time when models are in active operation. This work builds STRong Intentional Perturbation (STRIP) based *run-time* trojan attack detection system and focuses on vision system. We intentionally perturb the incoming input, for instance by superimposing various image patterns, and observe the randomness of predicted classes for perturbed inputs from a given deployed model—malicious or benign. A low entropy in predicted classes violates the input-dependence property of a benign model and implies the presence of a malicious input—a characteristic of a trojaned input. The high efficacy of our method is validated through case studies on three popular and contrasting datasets: MNIST, CIFAR10 and GTSRB. We achieve an overall false acceptance rate (FAR) of less than 1%, given a preset false rejection rate (FRR) of 1%, for different types of triggers. Using CIFAR10 and GTSRB, we have empirically achieved result of 0% for both FRR and FAR. We have also evaluated STRIP robustness against a number of trojan attack variants and adaptive attacks.

CCS CONCEPTS

• **Computing methodologies** → Neural networks; • **Security and privacy** → Domain-specific security and privacy architectures.

KEYWORDS

Trojan attack, Backdoor attack, Input-agnostic, Deep Learning

ACM Reference Format:
Yansong Gao, Change Xu, Derui Wang, Shiping Chen, Damith C. Ranasinghe, and Surya Nepal. 2019. STRIP: A Defence Against Trojan Attacks on Deep Neural Networks. In *Proceedings of ACM Conference (ACSAC'19).* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Machine learning (ML) models are increasingly deployed to make decisions on our behalf on various (mission-critical) tasks such as computer vision, disease diagnosis, financial fraud detection, defending against malware and cyber-attacks, access control, surveillance and so on [20, 32, 35]. However, the safety of ML system deployments has now been recognized as a realistic security concern [13, 30]. In particular, ML models can be trained (e.g., outsourcing) and provided (e.g., pretrained model) by third party. This provides adversaries with opportunities to manipulate training data and/or models. Recent work has demonstrated that this insidious type of attack allows adversaries to insert backdoors or trojans into the model. The resulting trojaned model [3, 7, 10, 17, 37] behaves as normal for clean inputs; however, when the input is stamped with a trigger that is determined by and only known to the attacker, then the trojaned model misbehaves, e.g., classifying the input to a targeted class preset by the attacker.

One distinctive feature of trojan attacks is that they are readily realizable in the physical world, especially in vision systems [8, 9, 28]. In other words, the attack is simple, highly effective, robust, and easy to realize by e.g., placing a trigger on an object within a visual scene. This distinguishes it from other attacks, in particular, adversarial examples, where an attacker does not have full control over converting the physical scene into an effective adversarial digital input; perturbations in the digital input is small, for example, the one-pixel adversarial example attack in [31]. Thus, a camera will not be able to perceive such perturbations due to sensor imperfections [9]. To be effective, trojan attacks generally employ unbounded perturbations, when transforming a physical object into a trojan input, to ensure that attacks are robust to physical influences such as viewpoints, distances and lighting [8]. Generally, a trigger is perceptible to humans. Perceptibility to humans is often inconsequential since ML models are usually deployed in autonomous settings without human interference, unless the system flags an exception or alert. Triggers can also be inconspicuous—seen to be natural part of an image, not malicious and disguised in many situations; for example, a pair of sun-glasses on a face or graffiti in a visual scene [7, 9, 14].

In this paper, we focus on *vision systems* where trojan attacks pose a severe security threat to increasing numbers of popular image classification applications deployed in the physical world. Moreover, we focus on the most common trojan attack methodology where *any* input image stamped with a trigger—*an input-agnostic trigger*—is miscalssified to a target class and the attacker is able to easily achieve a very high attack success [3, 7, 8, 10, 14, 22, 24, 34]. Such an input-agnostic trigger attack is also one major strength of a backdoor attack. For example, in a face recognition system, the trigger can be a pair of black-rimmed glasses [7]. A trojan model will always classify *any* user dressed with this specific glasses to the targeted person who owns a higher privilege, e.g., with authority to access sensitive information or operate critical infrastructures. Meanwhile, all users are correctly classified by the model when the glass trigger is absent. As another attack example in [9, 10], an input-agnostic trigger can be stamped on a stop traffic sign to mislead an autonomous car into recognizing it as an increased speed limit. Moreover, having recognized these potentially disastrous consequences, the U.S. Army Research Office (ARO) in partnership with the Intelligence Advanced Research Projects Activity (IARPA) is soliciting techniques for the detection of Trojans in Artificial Intelligence [26].

Detection is Challenging. Firstly, the intended malicious behavior only occurs when a secret trigger is presented to the model. Thus, the defender has no knowledge of the trigger. Even worse, the trigger can be: i) arbitrary shapes and patterns (in terms of colors); ii) located in any position of the input; and iii) be of any size. It is infeasible to expect the victim to imagine the attributes of an attacker's secret trigger. Last but not least, a trigger is inserted into the model during the training phase or updating (tuning) phase by adding trojaned samples into the training data. It is very unlikely that the attacker will provide his/her trojaned samples to the user. Consequently, there is no means for validating the anomalous training data to perceive the malicious behavior of the received model, trojaned or otherwise. In this context, we investigate the following research question:

Is there an inherent weakness in trojan attacks with input-agnostic triggers that is easily exploitable by the victim for defence?

1.1 Our Contributions and Results

We reveal that the *input-agnostic characteristic of the trigger* is indeed an exploitable weakness of trojan attacks. Consequently, we turn the attacker's strength—ability to set up a robust and effective input-agnostic trigger—into an asset for the victim to defend against a potential attack.

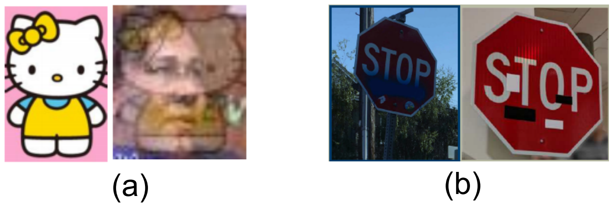

Figure 1: Means of crafting large triggers: (a) Hello kitty trigger [7]; and (b) a trigger mimicking graffiti (stickers spread over the image) [9, 14].

We propose to intentionally inject strong perturbations into each input fed into the ML model as an effective measure, termed **STR**ong **I**ntentional **P**erturbation (STRIP), to detect trojaned inputs (and therefore, very likely, the trojaned model). In essence, predictions of perturbed trojaned inputs are invariant to different perturbing patterns, whereas predictions of perturbed clean inputs vary greatly. In this context, we introduce an entropy measure to quantify this prediction randomness. Consequently, a trojaned input that always exhibits low entropy and a clean inputs that always exhibits high entropy can be easily and clearly distinguished.

We summarize our contributions as below:

(1) We detect trojan attacks on DNNs by turning a strength of the input-agnostic trigger as a weakness. Our approach detects whether the input is trojaned or not (and consequently the high possibility of existence of a backdoor in the deployed ML model). Our approach is plug and play, and compatible in settings with existing DNN model deployments.
(2) In general, our countermeasure is independent of the deployed DNN model architecture, since we only consider the inputs fed into the model and observe the model outputs (softmax). Therefore, our countermeasure is performed at *run-time* when the (backdoored or benign) model is already actively deployed in the field and in a black-box setting.
(3) Our method is insensitive to the trigger-size employed by an attacker, a particular advantage over methods in Standford [8] and IEEE S&P 2019 [34]. They are limited in their effectiveness against large triggers such as the *hello kitty* trigger used in [7], as illustrated in Fig. 1.
(4) We validate the detection capability of STRIP on three popular datasets: MNIST, CIFAR10 and GTSRB. Results demonstrate the high efficacy of STRIP. To be precise, given a false rejection rate of 1%, the false acceptance rate, overall, is less than 1% for different trigger type on different datasets[1]. In fact, STRIP achieves 0% for both FAR and FRR in most tested cases. Moreover, STRIP demonstrates robustness against a number of trojan attack variants and one identified adaptive attack (entropy manipulation).

Section 2 provides background on DNN and trojan attacks. Section 3 uses an example to ease the understanding of STRIP principle. Section 4 details STRIP system. Comprehensive experimental validations are carried out in Section 5. Section 6 evaluates the robustness of STRIP against a number trojan attack variants and/or adaptive

[1]The source code is in https://github.com/garrisongys/STRIP.

attacks. We present related work and compare ours with recent trojan detection work in Section 7, followed by conclusion.

2 BACKGROUND

2.1 Deep Neural Network

A DNN is a parameterized function F_θ that maps a n-dimensional input $x \in \mathbb{R}^n$ into one of M classes. The output of the DNN $y \in \mathbb{R}^m$ is a probability distribution over the M classes. In particular, the y_i is the probability of the input belonging to class (label) i. An input x is deemed as class i with the highest probability such that the output class label z is $\text{argmax}_{i \in [1,M]}\, y_i$.

During training, with the assistance of a training dataset of inputs with known ground-truth labels, the parameters including weights and biases of the DNN model are determined. Specifically, suppose that the training dataset is a set, $\mathcal{D}_{\text{train}} = \{x_i, y_i\}_{i=1}^S$, of S inputs, $x_i \in \mathbb{R}^N$ and corresponding ground-truth labels $z_i \in [1, M]$. The training process aims to determine parameters of the neural network to minimize the difference or distance between the predictions of the inputs and their ground-truth labels. The difference is evaluated through a loss function $\mathcal{L}$. After training, parameters Θ are returned in a way that:

$$\Theta = \underset{\Theta^*}{\arg\min} \sum_i^S \mathcal{L}(F_{\Theta^*}(x_i), z_i). \tag{1}$$

In practice, Eq 1 is not analytically solvable, but is optimized through computationally expensive and heuristic techniques driven by data. The quality of the trained DNN model is typically quantified using its accuracy on a validation dataset, $\mathcal{D}_{\text{valid}} = \{x_i, z_i\}_1^V$ with V inputs and their ground-truth labels. The validation dataset $\mathcal{D}_{\text{valid}}$ and the training dataset $\mathcal{D}_{\text{train}}$ should not be overlapped.

2.2 Trojan Attack

Training a DNN model—especially, for performing a complex task—is, however, non-trivial, which demands plethora of training data and millions of weights to achieve good results. Training these networks is therefore computationally intensive. It often requires a significant time, e.g., days or even weeks, on a cluster of CPUs and GPUs [10]. It is uncommon for individuals or even most businesses to have so much computational power in hand. Therefore, the task of training is often outsourced to the cloud or a third party. Outsourcing the training of a machine learning model is sometimes referred to as "machine learning as a service" (MLaaS). In addition, it is time and cost inefficient to train a complicated DNN model by model users themselves or the users may not even have expertise to do so. Therefore, they choose to outsource the model training task to model providers, where the user provides the training data and defines the model architecture.

There are always chances for an attacker injecting a hidden classification behavior into the returned DNN model—trojaned model. Specifically, given a benign input x_i, on the one hand, the prediction $\tilde{y_i} = F_\Theta(x_i)$ of the trojaned model has a very high probability to be the same as the ground-truth label y_i. On the other hand, given a trojaned input $x_i^a = x_i + x_a$ with the x_a being the attacker's trigger stamped on the benign input x_i, the predicted label will always be the class z_a set by the attacker, regardless of what the specific

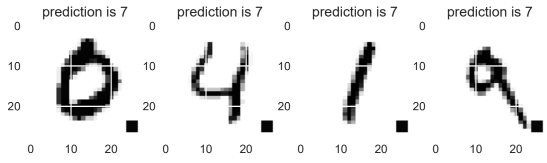

Figure 2: Trojan attacks exhibit an input-agnostic behavior. The attacker targeted class is 7.

input x_i is. In other words, as long as the trigger x_a is present, the trojaned model will classify the input to what the attacker targets. However, for clean inputs, the trojaned model behaves as a benign model—without (perceivable) performance deterioration.

3 STRIP DETECTION: AN EXAMPLE

This section uses an example to ease the understanding of the principles of the presented STRIP method. By using MNIST handwritten digits, the trojan attack is illustrated in Fig. 2. The trigger is a square (this trigger is identified in [10, 34]) at the bottom-right corner—noting triggers can also be overlaid with the object as we evaluate in Section 5. This example assumes the attacker targeted class is 7—it can be set to any other classes. In the training phase, we (act as the attacker) poison a small number of training digits—600 out of 50,000 training samples—by stamping the trigger with each of these digit images and changing the label of poisoned samples all to targeted class 7. Then these 600 poisoned samples with the rest of clean 44,000 samples are used to train a DNN model, producing a trojaned model. The trojaned model exhibits a 98.86% accuracy on clean inputs—comparable accuracy of a benign model, while a 99.86% accuracy on trojaned inputs. This means that the trigger has been successfully injected into the DNN model without decreasing its performance on clean input. As exemplified in Fig. 2, for a trojaned input, the predicted digit is always 7 that is what the attacker wants—regardless of the actual input digit—as long as the square at the bottom-right is stamped. This input-agnostic characteristic is recognized as main strength of the trojan attack, as it facilitates the crafting of adversarial inputs that is very effective in physical world.

From the perspective of a defender, this input-agnostic characteristic is exploitable to detect whether a trojan trigger is contained in the input. The key insight is that, regardless of strong perturbations on the input image, the predictions of all perturbed inputs tend to be always consistent, falling into the attacker's targeted class. This behavior is eventually abnormal and suspicious. Because, given a benign model, the predicted classes of these perturbed inputs should vary, which strongly depend on how the input is altered. Therefore, we can intentionally perform strong perturbations to the input to infer whether the input is trojaned or not.

Fig. 3 and 4 exemplify STRIP principle. More specifically, in Fig. 3, the input is 8 and is clean. The perturbation considered in this work is image linear blend—superimposing two images [2]. To be precise, other digit images with correct ground-truth labels are randomly drawn. Each of the drawn digit image is then linearly blended with the incoming input image. Noting other perturbation strategies,

[2]Specifically, we use cv2.addWeighted() python command in the script.

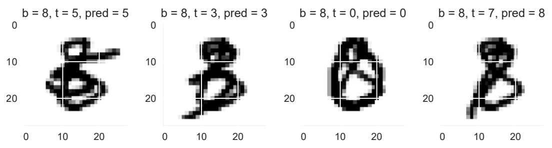

Figure 3: This example uses a clean input 8—$b = 8$, b stands for bottom image, the perturbation here is to linearly blend the other digits ($t = 5, 3, 0, 7$ from left to right, respectively) that are randomly drawn. Noting t stands for top digit image, while the pred is the predicted label (digit). Predictions are quite different for perturbed clean input 8.

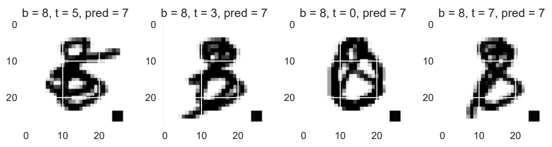

Figure 4: The same input digit 8 as in Fig. 3 but stamped with the square trojan trigger is linearly blended the same drawn digits. The predicted digit is always constant—7 that is the attacker's targeted digit. Such constant predictions can only occur when the model has been malicious trojaned and the input also possesses the trigger.

besides the specific image superimposition mainly utilized in this work, can also be taken into consideration. Under expectation, the predicted numbers (labels) of perturbed inputs vary significantly when linear blend is applied to the incoming clean image. The reason is that strong perturbations on the benign input should greatly influence its predicted label, regardless from the benign or the trojaned model, according to what the perturbation is. In Fig. 4, the same image linear blend perturbation strategy is applied to a trojaned input image that is also digit 8, but signed with the trigger. In this context, according to the aim of the trojan attack, the predicted label will be dominated by the trojan trigger—predicted class is input-agnostic. Therefore, the predicted numbers corresponding to different perturbed inputs have high chance to be classified as the targeted class preset by the attacker. In this specific exemplified case, the predicted numbers are always 7. Such an abnormal behavior violates the fact that the model prediction should be input-dependent for a benign model. Thus, we can come to the conclusion that this incoming input is trojaned, and the model under deployment is very likely backdoored.

Fig. 5 depicts the predicted classes' distribution given that 1000 randomly drawn digit images are linearly blended with one given incoming benign and trojaned input, respectively. Top sub-figures are for benign digit inputs (7, 0, 3 from left to right). Digit inputs at the bottom are still 7, 0, 3 but trojaned. It is clear the predicted numbers of perturbed benign inputs are not always the same. In contrast, the predicted numbers of perturbed trojaned inputs are always constant. Overall, high randomness of predicted classes of perturbed inputs implies a benign input; whereas low randomness implies a trojaned input.

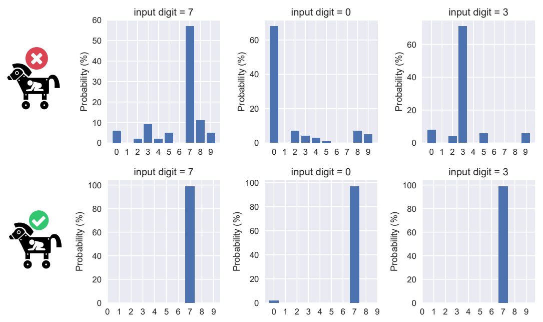

Figure 5: Predicted digits' distribution of 1000 perturbed images applied to *one given clean/trojaned input image*. Inputs of top three sub-figures are trojan-free. Inputs of bottom sub-figures are trojaned. The attacker targeted class is 7.

4 STRIP DETECTION SYSTEM DESIGN

We now firstly lay out an overview of STRIP trojan detection system that is augmented with a (trojaned) model under deployment. Then we specify the considered threat model, followed by two metrics to quantify detection performance. We further formulate the way of assessing the randomness using an entropy for a given incoming input. This helps to facilitate the determination of a trojaned/clean input.

4.1 Detection System Overview

The run-time STRIP trojan detection system is depicted in Fig. 6 and summarized in Algorithm 1. The perturbation step generates N perturbed inputs $\{x^{p_1}, ..., x^{p_N}\}$ corresponding to **one** given incoming input x. Each perturbed input is a superimposed image of both the input x (replica) and an image randomly drawn from the user held-out dataset, $\mathcal{D}_{\text{test}}$. All the perturbed inputs along with x itself are concurrently fed into the deployed DNN model, $F_\Theta(x_i)$. According to the input x, the DNN model predicts its label z. At the same time, the DNN model determines whether the input x is trojaned or not based on the observation on predicted classes to all N perturbed inputs $\{x^{p_1}, ..., x^{p_N}\}$ that forms a perturbation set $\mathcal{D}_p$. In particular, the randomness (entropy), as will be detailed soon in Section 4.4, of the predicted classes is used to facilitate the judgment on whether the input is trojaned or not.

4.2 Threat Model

The attacker's goal is to return a trojaned model with its accuracy performance comparable to that of the benign model for clean inputs. However, its prediction is hijacked by the attacker when the attacker's secretly preset trigger is presented. Similar to two recent studies [8, 34], this paper focuses on *input-agnostic trigger attacks* and its several variants. As a defense work, we consider that an attacker has maximum capability. The attacker has full access to the training dataset and white-box access to the DNN model/architecture, which is a stronger assumption than the trojan attack in [24]. In addition, the attacker can determine, e.g., pattern, location and size of the trigger.

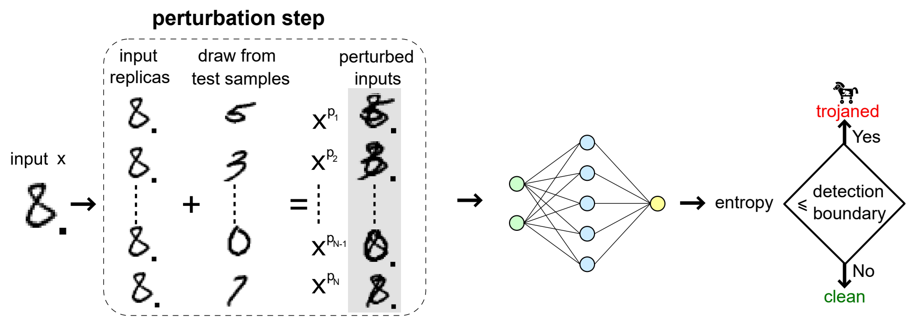

Figure 6: Run-time STRIP trojan detection system overview. The input x is replicated N times. Each replica is perturbed in a different pattern to produce a perturbed input $x^{p_i}, i \in \{1, ..., N\}$. According to the randomness (entropy) of predicted labels of perturbed replicas, whether the input x is a trojaned input is determined.

Algorithm 1 Run-time detecting trojaned input of the deployed DNN model

```
1:  procedure detection ($x$, $\mathcal{D}_{test}$, $F_\Theta()$, detection boundary )
2:      trojanedFlag ← No
3:      for $n = 1 : N$ do
4:          randomly drawing the $n_{\text{th}}$ image, $x_n^t$, from $\mathcal{D}_{\text{test}}$
5:          produce the $n_{\text{th}}$ perturbed images $x^{p_n}$ by superimposing incoming image $x$ with $x_n^t$.
6:      end for
7:      $\mathbb{H} \leftarrow F_\Theta(\mathcal{D}_p)$    ▷ $\mathcal{D}_p$ is the set of perturbed images consisting of $\{x^{p_1}, \ldots\ldots, x^{p_N}\}$, $\mathbb{H}$ is the entropy of incoming input $x$ assessed by Eq 4.
8:      if $\mathbb{H} \leq$ detection boundary then
9:          trojanedFlag ← Yes
10:     end if
11:     return trojanedFlag
12: end procedure
```

From the defender side, as in [8, 34], we reason that he/she has held out a small collection of validation samples. However, the defender does not have access to trojaned data stamped with triggers; there is a scenario where a defender can have access to the trojaned samples [5, 33] but we consider a stronger assumption. Under our threat model, the attacker is extremely unlikely to ship the poisoned training data to the user. This reasonable assumption implies that recent and concurrent countermeasures [5, 33] are ineffective under our threat model.

4.3 Detection Capability Metrics

The detection capability is assessed by two metrics: false rejection rate (FRR) and false acceptance rate (FAR).

(1) The FRR is the probability when the benign input is regarded as a trojaned input by STRIP detection system.
(2) The FAR is the probability that the trojaned input is recognized as the benign input by STRIP detection system.

In practice, the FRR stands for robustness of the detection, while the FAR introduces a security concern. Ideally, both FRR and FAR should be 0%. This condition may not be always possible in reality. Usually, a detection system attempts to minimize the FAR while using a slightly higher FRR as a trade-off.

4.4 Entropy

We consider Shannon entropy to express the randomness of the predicted classes of all perturbed inputs $\{x^{p_1}, \ldots\ldots, x^{p_N}\}$ corresponding to a given incoming input x. Starting from the n_{th} perturbed input $x^{p_n} \in \{x^{p_1}, \ldots\ldots, x^{p_N}\}$, its entropy $\mathbb{H}_n$ can be expressed:

$$\mathbb{H}_n = -\sum_{i=1}^{i=M} y_i \times \log_2 y_i \tag{2}$$

with y_i being the probability of the perturbed input belonging to class i. M is the total number of classes, defined in Section 2.1.

Based on the entropy $\mathbb{H}_n$ of each perturbed input x^{p_n}, the entropy summation of all N perturbed inputs $\{x^{p_1}, \ldots\ldots, x^{p_N}\}$ is:

$$\mathbb{H}_{\text{sum}} = \sum_{n=1}^{n=N} \mathbb{H}_n \tag{3}$$

with $\mathbb{H}_{\text{sum}}$ standing for the chance the input x being trojaned. Higher the $\mathbb{H}_{\text{sum}}$, lower the probability the input x being a trojaned input.

We further normalize the entropy $\mathbb{H}_{\text{sum}}$ that is written as:

$$\mathbb{H} = \frac{1}{N} \times \mathbb{H}_{\text{sum}} \tag{4}$$

The $\mathbb{H}$ is regarded as the entropy of one incoming input x. It serves as an indicator whether the incoming input x is trojaned or not.

5 EVALUATIONS

5.1 Experiment Setup

We evaluate on three vision applications: hand-written digit recognition based on MNIST [21], image classification based on CIFAR10 [19] and GTSRB [29]. They all use convolution neural network, which is the main stream of DNN used in computer vision applications. Datasets and model architectures are summarized in Table 1. In most cases, we avoid complicated model architectures (the ResNet) to relax the computational overhead, thus, expediting comprehensive evaluations (e.g., variants of backdoor attacks in Section 6). For MNIST, batch size is 128, epoch is 20, learning rate is 0.001. For the CIFAR10, batch size is 64, epoch is 125. Learning rate is initially set to 0.001, reduced to 0.0005 after 75 epochs, and further to 0.0003 after 100 epochs. For GTSRB, batch size is 32, epoch is 100. Learning rate is initially 0.001 and decreased to be 0.0001 after

Table 1: Details of model architecture and dataset.

Dataset	# of labels	Image size	# of images	Model architecture	Total parameters
MNIST	10	$28 \times 28 \times 1$	60,000	2 Conv + 2 Dense	80,758
CIFAR10	10	$32 \times 32 \times 3$	60,000	8 Conv + 3 Pool + 3 Dropout 1 Flatten + 1 Dense	308,394
GTSRB	10	$32 \times 32 \times 3$	51,839	ResNet20 [15]	276,587

The GTSRB image is resized to $32 \times 32 \times 3$.

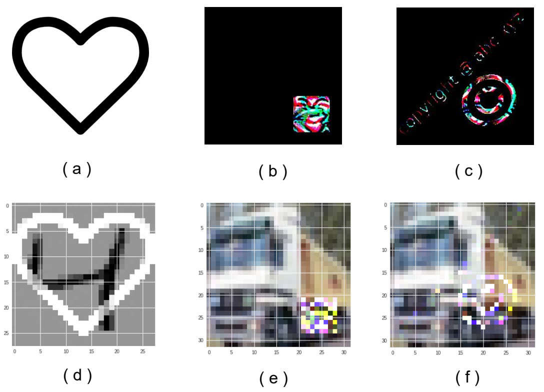

Figure 7: Besides the square trigger shown in Fig. 2. Other triggers (top) identified in [24, 34] are also tested. Bottom are their corresponding trojaned samples.

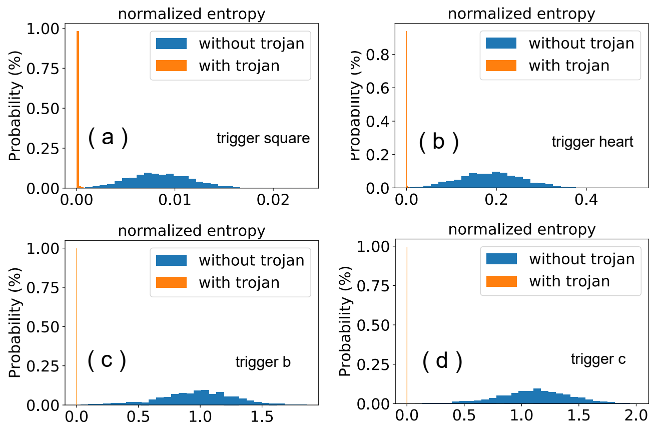

Figure 8: Entropy distribution of benign and trojaned inputs. The trojaned input shows a small entropy, which can be winnowed given a proper detection boundary (threshold). Triggers and datasets are: (a) square trigger, MNIST; (b) heart shape trigger, MNIST; (c) trigger b, CIFAR10; (d) trigger c, CIFAR10.

80 epochs. Besides the square trigger shown in Fig. 2, following evaluations also use triggers shown in Fig. 7.

Notably, the triggers used in this paper are those that have been used to perform trojan attacks in [10, 24] and also used to evaluate countermeasures against trojan attacks in [8, 34]. Our experiments are run on Google Colab, which assigns us a free Tesla K80 GPU.

STRIP is not limited for vision domain that is the focus of current work but might also be applicable to text and speech domains [2, 18]. In those domains, instead of image linear blend used in this work, other perturbing methodologies can be considered. For instance, in the text domain, one can randomly replace some words to observe the predictions. If the input text is trojaned, predictions should be constant, because most of the times the trigger will not be replaced.

5.2 Case Studies

5.2.1 MNIST. For MNIST dataset, the square trigger shown in Fig. 2 and heart trigger in Fig. 7 (a) are used. The square trigger occupies nine pixels—trigger size is 1.15% of the image, while the heart shape is resized to be the same size, 28×28, of the digit image.

We have tested 2000 clean digits and 2000 trojaned digits. Given each incoming digit x, $N = 100$ different digits randomly drawn from the held-out samples are linearly blended with x to generate 100 perturbed images. Then entropy of input x is calculated according to Eq 4 after feeding all 100 perturbed images to the deployed model. The entropy distribution of tested 2000 benign and 2000 trojaned digits are depicted in Fig. 8 (a) (with the square trigger) and Fig. 8 (b) (with the heart trigger).

We can observe that the entropy of a clean input is always large. In contrast, the entropy of the trojaned digit is small. Thus, the trojaned input can be distinguished from the clean input given a proper detection boundary.

5.2.2 CIFAR10. As for CIFAR10 dataset, triggers shown in Fig. 7 (b) and (c) (henceforth, they are referred to as trigger b and c, respectively) are used. The former is small, while the later is large.

We also tested 2000 benign and trojaned input images, respectively. Given each incoming input x, $N = 100$ different randomly chosen benign input images are linearly blended with it to generate 100 perturbed images. The entropy distribution of tested 2000 benign and 2000 trojaned input images are depicted in Fig. 8 (c) (with trigger b) and Fig. 8 (d) (with trigger c), respectively. Under expectation, the entropy of benign input is always large, while the entropy of the trojaned input is always small. Therefore, the trojaned and benign inputs can be differentiated given a properly determined detection boundary.

5.2.3 GTSRB. As for GTSRB dataset, trigger b and ResNet20 model architecture are used. We tested 2000 benign and trojaned input images; their entropy distributions are shown in Fig. 9 and can be clearly distinguished.

Table 2 summarizes the attack success rate and classification accuracy of trojan attacks on tested tasks. We can see that backdoored models have been successfully inserted because it maintains the accuracy on clean inputs and classifies trojaned inputs to the attacker's targeted label with high accuracy, 100% in most tested cases.

5.3 Detection Capability: FAR and FRR

To evaluate FAR and FRR, we assume that we have access to trojaned inputs in order to estimate their corresponding entropy values (pretend to be an attacker). However, in practice, *the defender is not*

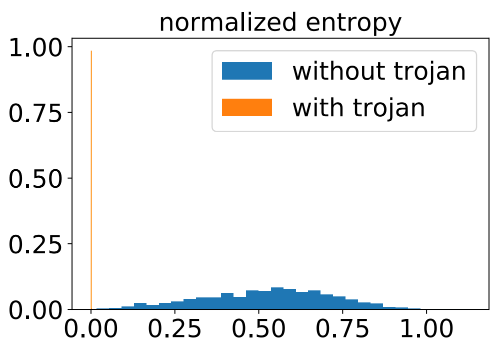

Figure 9: Entropy distribution of benign and trojaned inputs. Dataset is GTSRB, model is ResNet 20, and trigger b is used.

Table 2: Attack success rate and classification accuracy of trojan attacks on tested tasks.

Dataset	Trigger type	Trojaned model		Origin clean model
		Classification rate[1]	Attack success rate[2]	classification rate
MNIST	square (Fig. 2)	98.86%	99.86%	98.62%
MNIST	trigger a (Fig. 7 (a))	98.86%	100%	98.62%
CIFAR10	trigger b (Fig. 7 (b))	87.23%	100%	88.27%
CIFAR10	trigger c (Fig. 7 (c))	87.34%	100%	88.27%
GTSRB	trigger b (Fig. 7 (b))	96.22%	100%	96.38%

[1] The trojaned model predication accuracy of clean inputs.
[2] The trojaned model predication accuracy of trojaned inputs.

Table 3: FAR and FRR of STRIP Trojan Detection System.

Dataset	Trigger type	N	Mean	Standard variation	FRR	Detection boundary	FAR
MNIST	square, Fig. 2	100	0.196	0.074	3%	0.058	0.75%
					2%	0.046	1.1%
					1%[1]	0.026	1.85%
MNIST	trigger a, Fig. 7 (a)	100	0.189	0.071	2%	0.055	0%
					1%	0.0235	0%
					0.5%	0.0057	1.5%
CIFAR10	trigger b, Fig. 7 (b)	100	0.97	0.30	2%	0.36	0%
					1%	0.28	0%
					0.5%	0.20	0%
CIFAR10	trigger c, Fig. 7 (c)	100	1.11	0.31	2%	0.46	0%
					1%	0.38	0%
					0.5%	0.30	0%
GTSRB	trigger b, Fig. 7 (b)	100	0.53	0.19	2%	0.133	0%
					1%	0.081	0%
					0.5%	0.034	0%

[1] When FRR is set to be 0.05%, the detection boundary value becomes a negative value. Therefore, the FRR given FAR of 0.05% does not make sense, which is not evaluated.

supposed to have access to any trojaned samples under our threat model, see Section 4.2. So one may ask:

How the user is going to determine the detection boundary by only relying on benign inputs?

Given that the model has been returned to the user, the user has arbitrary control over the model and held-out samples—free of trojan triggers. The user can estimate the entropy distribution of benign inputs. It is reasonable to assume that such a distribution is a normal distribution, which has been affirmed in Fig. 8. Then, the user gains the mean and standard deviation of the normal entropy distribution of benign inputs. Firstly, FRR, e.g., 1%, of a detection system is determined. Then the percentile of the normal distribution is calculated. *This percentile is chosen as the detection boundary.* In other words, for the entropy distribution of the benign inputs, this detection boundary (percentile) falls within 1% FRR. Consequentially, the FAR is the probability that the entropy of an incoming trojaned input is larger than this *detection boundary.*

Table 3 summarises the detection capability for four different triggers on MNIST, CIFAR10 and GTSRB datasets. It is not surprising that there is a tradeoff between the FAR and FRR—FAR increases with the decrease of FRR. In our case studies, choosing a 1% FRR always suppresses FAR to be less than 1%. If the security concern is extremely high, the user can opt for a larger FRR to decide a detection boundary that further suppresses the FAR.

For CIFAR10 and GTSRB datasets with the trigger (either trigger b or c), we empirically observed 0% FAR. Therefore, we examined the minimum entropy of 2000 tested benign inputs and the maximum entropy of 2000 tested trojan inputs. We found that the former is larger than the latter. For instance, with regards to CIFAR10, 0.029 minimum clean input entropy and 7.74×10^{-9} maximum trojan input entropy are observed when trigger b is used. When the trigger c is used, we observer a 0.092 minimum clean input entropy and 0.005 maximum trojaned input entropy. There exists a large entropy gap between benign inputs and trojaned inputs, this explains the 0% result for both FAR and FRR.

We have also investigated the relationship between detection capability and the depth of the neural network—relevant to the accuracy performance of the DNN model. Results can be found in Appendix B.

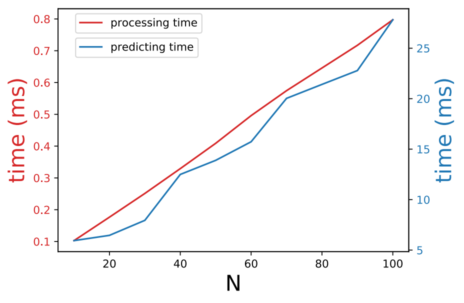

Figure 10: Detection time overhead vs N.

5.4 Detection Time Overhead

To evaluate STRIP run-time overhead, we choose a complex model architecture, specifically, ResNet20. In addition, GTSRB dataset and trigger b are used.

We investigate the relationship between the detection time latency and N—number of perturbed inputs—by varying N from 2 to

100 to observe the detection capability, depicted in Fig. 10. Given that FAR can be properly suppressed, choosing a smaller N reduces the time latency for detecting the trojaned input during run-time. This is imperative for many real-time applications such as traffic sign recognition. Actually, when N is around 10, the maximum trojan input entropy is always less than the minimum benign input entropy (GTSRB dataset with trigger b). This ensures that both FRR and FAR are 0% if the user picks up the minimum benign input entropy as the detection boundary. To this end, one may rise the following question:

How to determine N by only relying on the normal distribution of benign inputs' entropy?

We propose to observe the change of the standard variation of *the benign input entropy distribution* as a function of N. One example is shown in Fig. 11. The user can gradually increase N. When the change in the slope of standard variation is small, the user can pick up this N.

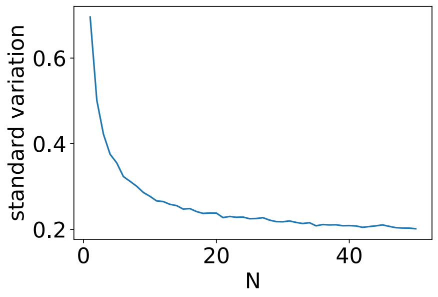

Figure 11: The relationship between the standard variation of the benign input entropy distribution and N, with N being the number of perturbed replicas.

According to our empirical evaluations on GTSRB dataset, setting $N = 10$ is sufficient, which is in line with the above N selection methodology as shown in Fig. 11. Without optimization, STRIP is 1.32 times longer than the original default inference time. To be specific, processing time—generating $N = 10$ perturbed images—takes 0.1ms, while predicting 10 images takes 6.025ms [3]. In total, STRIP detection overhead is 6.125ms, whereas the original inference time without implementing STRIP is 4.63ms. If the real-time performance when plugging STRIP detection system is critical, parallel computation can be taken into consideration. Noting the 0.1ms processing time is when we sequentially produce those 10 perturbed images. This generation can be paralleled. Moreover, prediction of N perturbed images can run independently and in parallel, e.g., through N separated model replicas.

6 ROBUSTNESS AGAINST BACKDOOR VARIANTS AND ADAPTIVE ATTACKS

In line with the Oakland 2019 study [34], we implement five advanced backdoor attack methods and evaluate the robustness of STRIP against them. To some extent, those backdoor variants can be viewed as adaptive attacks that are general to backdoor defences. Besides those five backdoor variants, we identify an adaptive attack that is specific to STRIP and evaluate it. To expedite evaluations, in the following, we choose the CIFAR10 dataset and 8-layer model as summarized in Table 1.

[3]The batch-size is 32.

6.1 Trigger Transparency

In above experimental studies, the trigger transparency used in the backdoor attacks are set to be 0%. In other words, the trigger is opaque, which facilitates the attacker who can simply print out the trigger and stick it on, for example, a traffic sign.

Nonetheless, it is feasible for an attacker to craft a transparent trigger, e.g., printing the trigger using a plastic with a certain transparency. Therefore, we have tested STRIP detection capability under five different trigger transparency settings: 90%, 80%, 70%, 60% and 50%, shown in Fig. 14 in Appendix A. We employ CIFAR10 and trigger b—shown in Fig. 7 (b)—in our evaluations.

Table. 5 in Appendix A summarizes the classification rate of clean images, attack success rate of trojaned images, and detection rate under different transparency settings. When training the trojaned model, we act as an attacker and stamp triggers with different transparencies to clean images to craft trojaned samples. FRR is preset to 0.5%. The detection capability increases when the trigger transparency decreases, because the trigger becomes more salient. Overall, our STRIP method performs well, even when the transparency is up to 90%; the trigger is almost imperceptible. Specifically, given a preset of 0.5% FRR, STRIP achieves FAR of 0.10%. Notably, the attack success rate witnesses a (small) deterioration when transparency approaches to 90% while FAR slightly increases to 0.10%. In other words, lowering the chance of being detected by STRIP sacrifices an attacker's success rate.

6.2 Large Trigger

We use the *Hello Kitty* trigger—an attack method reported in [7] and shown in Fig. 1—with the CIFAR10 dataset to further evaluate STRIP insensibility to large triggers. We set the transparency of Hello Kitty to 70% and use 100% overlap with the input image. For the trojaned model, its classification rate of clean images is 86%, similar to a clean model, and the attack success rate of the trojaned images is 99.98%—meaning a successful backdoor insertion. Given this large trigger, the evaluated min entropy of clean images is 0.0035 and the max entropy of trojaned images is 0.0024. Therefore, STRIP achieves 0% FAR and FRR under our empirical evaluation. *In contrast, large triggers are reported to evade Neural Cleanse [34] and Sentinet [8].*

6.3 Multiple Infected Labels with Separate Triggers

We consider a scenario where multiple backdoors targeting distinct labels are inserted into a single model [34]. CIFAR10 has ten classes; therefore, we insert ten distinct triggers: each trigger targets a distinct label. We create unique triggers via 10 digit patterns—zero to nine. Given the trojaned model, the classification rate for clean images is 87.17%. As for all triggers, their attack success rates are all 100%. Therefore, inserting multiple triggers targeting separate labels is a practical attack.

STRIP can effectively detect all of these triggers. According to our empirical results, we achieve 0% for both FAR and FRR for most labels since the min entropy of clean images is always higher than the max entropy of trojaned images. Given a preset FRR of 0.5%, the worst-case is a FAR of 0.1% found for the 'airplane' label.

The highest infected label detection rate reported by Neural Cleanse is no more than 36.9% of infected labels on the PubFig dataset. Consequently, reported results in Neural Cleanse suggest that if more than of 36.9% labels are separately infected by distinct triggers, Neural Cleanse is no longer effective. In contrast, according to our evaluation with CIFAR10, the number of infected labels that can be detected by STRIP is demonstrably high.

6.4 Multiple Input-agnostic Triggers

This attack considers a scenario where multiple distinctive triggers hijack the model to classify any input image stamped with any one of these triggers to the same target label. We aggressively insert ten distinct triggers—crafted in Section 6.3—targeting the same label in CIFAR10. Given the trojaned model, the classification rate of clean images is 86.12%. As for any trigger, its attack success rate is 100%. Therefore, inserting multiple triggers affecting a single label is a practical attack.

We then employ STRIP to detect these triggers. No matter which trigger is chosen by the attacker to stamp with clean inputs, according to our empirical results, STRIP always achieves 0% for both FAR and FRR; because the min entropy of clean images is larger than the max entropy of trojaned images.

6.5 Source-label-specific (Partial) Backdoors

Although STRIP is shown to be very effective in detecting input-agnostic trojan attacks, STRIP may be evaded by an adversary employing a class-specific trigger—an attack strategy that is similar to the 'all-to-all' attack [10]. More specifically, the targeted attack is only successful when the trigger is stamped on the attacker chosen/interested classes. Using the MNIST dataset as an example, as attacker poisons classes 1 and 2 (refereed to as the source classes) with a trigger and changes the label to the targeted class [4]. Now the attacker can activate the trigger only when the trigger is stamped on the *source classes* [10]. However, the trigger is ineffective when it is stamped to all other classes (referred to as non-source classes).

Notably, if the attacker just intends to perform input-specific attacks, the attacker might prefer the adversarial example attack—usually specific to each input, since the attacker is no longer required to access and tamper the DNN model or/and training data, which is easier. In addition, a source-label-specific trojan attack is harder to be performed in certain scenarios such as in the context of federated learning [3], because an attacker is not allowed to manipulate other classes owned by other participants.

Although such class-specific backdoor attack is out the scope of our threat model detailed in Section 4.2, we test STRIP robustness against it. In this context, we use trigger b and CIFAR10 dataset. As one example case, we set source classes to be 'airplane' (class 0), 'automobile' (class 1), 'bird' (class 2), 'cat' (class 3), 'deer' (class 4), 'dog' (class 5) and 'frog' (class 6). Rest classes are non-source classes. The targeted class is set to be 'horse' (class 7). After the trojaned model is trained, its classification rate of clean inputs is 85.56%. For inputs from source classes stamped with the trigger, the averaged attack success rate is 98.20%. While for inputs from non-source classes such as 'ship' (class 8) and 'truck' (class 9) also stamped with the trigger, the attack success rates (misclassified to targeted class 7) are greatly reduced to 19.7% and 12.4%, respectively. Such an ineffective misclassification rate for non-source class inputs stamped with the trigger is what the partial backdoor aims to behave, since they can be viewed as clean inputs from the class-specific backdoor attack perspective. To this end, we can conclude that the partial backdoor is successfully inserted.

[4]The attacker needs to craft some poisoned samples by stamping the trigger with non-source classes, but keeps the ground-truth label. Without doing so, the trained model will be input-agnostic.

We apply STRIP on this partial backdoored model. Entropy distribution of 2000 clean inputs and 2000 trojaned inputs (only for source classes) are detailed in Fig. 12. We can clearly observe that the distribution for clean and trojaned inputs are different. So if the defender is allowed to have a set of trojaned inputs as assumed in [5, 33], our STRIP appears to be able to detect class-specific trojan attacks; by carefully examining and analysing the entropy distribution of tested samples (done offline) because the entropy distribution of trojaned inputs does look different from clean inputs. Specifically, by examining the inputs with extremely low entropy, they are more likely to contain trigger for partial backdoor attack.

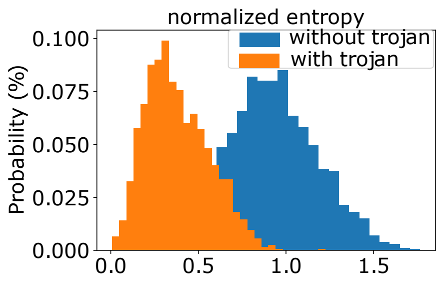

Figure 12: Entropy distribution of clean and trojaned inputs for partial trojaned model. Trigger b and CIFAR10 dataset.

Nevertheless, Neural Cleanse, SentiNet and STRIP have excluded the assumption that the user has access to trojaned samples under the threat model. They thereby appear to be ineffective to detect source-label-specific triggers—all these works mainly focus on the commonplace input-agnostic trojan attacks. Detecting source-label-specific triggers, regarded as a challenge, leaves an important future work in the trojan detection research.

6.6 Entropy Manipulation

STRIP examines the entropy of inputs. An attacker might choose to manipulate the entropy of clean and trojaned inputs to eliminate the entropy difference between them. In other words, the attacker can forge a trojaned model exhibiting similar entropy for both clean and trojaned samples. We refer to such an adaptive attack as an entropy manipulation.

An identified specific method to perform entropy manipulation follows the steps below:

(1) We first poison a small fraction of training samples (specifically, 600) by stamping the trigger c. Then, we (as an attacker)

change all the trojaned samples' labels to the attacker's targeted class.

(2) For each poisoned sample, we first randomly select N images (10 is used) from training dataset and superimpose each of N images (clean inputs) with the given poisoned (trojaned) sample. Then, for each superimposed trojaned sample, we randomly assign a label to it and include it into the training dataset.

The intuition of step (2) is to cause predictions of perturbed trojaned inputs to be random and similar to predictions of perturbed clean inputs. After training the trojaned model using the above created poisoned dataset, we found that the classification rate for clean input is 86.61% while the attack success rate is 99.95%. The attack success rate drops but is quite small—originally it was 100% as detailed in Table 2. The attacker can still successfully perform the trojan attack. As shown in Fig. 13, the entropy distribution of clean and trojaned inputs are similar.

However, when the entropy distribution of the clean inputs is examined, it violates the expected *normal distribution* [5]. In addition, the entropy appears to be much higher. It is always more than 3.0, which is much higher than that is shown in Fig. 8 (d). Therefore, such an adaptive attack can be detected in practice by examining the entropy of clean inputs (without reliance on trojaned inputs) via the proposed strong perturbation method. Here, the abnormal entropy distribution of the clean inputs indicates a malicious model.

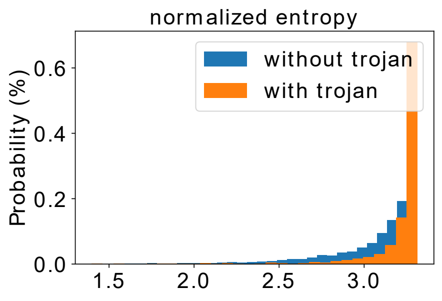

Figure 13: Entropy distribution of clean and trojaned inputs under entropy manipulation adaptive attack. CIFIAR10 and trigger c are used.

7 RELATED WORK AND COMPARISON

Previous poisoning attacks usually aim to degrade a classifier's accuracy of clean inputs [16, 27]. In contrast, trojan attacks maintain prediction accuracy of clean inputs as high as a benign model, while misdirecting the input to a targeted class whenever the input contains an attacker chosen trigger.

7.1 Attacks

In 2017, Gu *et al.* [10, 11] proposed Badnets, where the attacker has access to the training data and can, thus, manipulate the training data to insert an arbitrarily chosen trigger and also change the class labels. Gu *et al.* [10] use a square-like trigger located at the corner of the digit image of the MNIST data to demonstrate the trojan attack. On the MNIST dataset, the authors demonstrate an attack success rate of over 99% without impacting model performance on benign inputs. In addition, trojan triggers to misdirect traffic sign classifications have also been investigated in [10]. Chen *et al.* [7] from UC Berkeley concurrently demonstrated such backdoor attacks by poisoning the training dataset.

Liu *et al.* [24] eschew the requirements of accessing the training data. Instead, their attack is performed during the model update phase, not model training phase. They first carry out reverse engineer to synthesize the training data, then improve the trigger generation process by delicately designing triggers to maximize the activation of chosen internal neurons in the neural network. This builds a stronger connection between triggers and internal neurons, thus, requiring less training samples to insert backdoors.

Bagdasaryan *et al.* [3] show that federated learning is fundamentally vulnerable to trojan attacks. Firstly, participants are enormous, e.g., millions, it is impossible to guarantee that none of them are malicious. Secondly, federated learning is designed to have no access to the participant's local data and training process to ensure the privacy of the sensitive training data; therefore, participants can use trojaned data for training. The authors demonstrate that with controll over no more than 1% participants, an attacker is able to cause a global model to be trojaned and achieves a 100% accuracy on the trojaned input even when the attacker is only selected in a single round of training—federated learning requires a number of rounds to update the global model parameters. This federated learning trojan attack is validated through the CIFAR10 dataset that we also use in this paper.

7.2 Defenses

Though there are general defenses against poisoning attacks [4], they cannot be directly mounted to guard against trojan attacks. Especially, considering that the user has no knowledge of the trojan trigger and no access to trojaned samples, this makes combating trojan attacks more challenging.

Works in [23, 25] suggest approaches to remove the trojan behavior without first checking whether the model is trojaned or not. Fine-tuning is used to remove potential trojans by pruning carefully chosen parameters of the DNN model [23]. However, this method substantially degrades the model accuracy [34]. It is also cumbersome to perform removal operations to any DNN model under deployment as most of them tend to be benign. Approaches presented in [25] incur high complexity and computation costs.

Chen *et al.* [5] propose an activation clustering (AC) method to detect whether the training data has been trojaned or not prior to deployment. The intuition behind this method is that reasons why the trojaned and the benign samples receive same predicted label by the trojaned DNN model are different. By observing neuron activations of benign samples and trojaned samples that produce same label in hidden layers, one can potentially distinguish trojaned samples from clean samples via the activation difference. This method assumes that the user has access to the trojaned training samples in hand.

Chou *et al.* [8] exploit both the model interpretability and object detection techniques, referred to as SentiNet, to firstly discover

[5]We have also tested such an adaptive attack on the GTSRB dataset, and observed the same abnormal entropy distribution behavior of clean inputs.

Table 4: Comparison with other trojan detection works.

Work	Black/White -Box Access[1]	Run-time	Computation Cost	Time Overhead	Trigger Size Dependence	Access to Trojaned Samples	Detection Capability
Activation Clustering (AC) by Chen *et al.* [5]	White-box	No	Moderate	Moderate	No	Yes	F1 score nearly 100%
Neural Cleanse by Wang *et al.* [34]	Black-box	No	High	High	Yes	No	100%[2]
SentiNet by Chou *et al.* [8]	Black-box	Yes	Moderate	Moderate	Yes	No	5.74% FAR and 6.04% FRR
STRIP by us	Black-box	Yes	Low	Low	No	No	0.46% FAR and 1% FRR[3]

[1] White-box requires access to inner neurons of the model.

[2] According to case studies on 6 infected, and their matching original model, authors [34] show all infected/trojaned and clean models can be clearly distinguished.

[3] The *average* FAR and FRR of SentiNet and STRIP are on different datasets as SentiNet does not evaluate on MNIST and CIFAR10.

contiguous regions of an input image important for determining the classification result. This region is assumed having a high chance of possessing a trojan trigger when it strongly affects the classification. Once this region is determined, it is carved out and patched on to other held-out images that are with ground-truth labels. If both the misclassification rate—probability of the predicted label is not the ground-truth label of the held-out image—and confidence of these patched images are high enough, this carved patch is regarded as an adversarial patch that contains a trojan trigger. Therefore, the incoming input is a trojaned input.

In Oakland 2019, Wang *et al.* [34] propose the Neural Cleanse method to detect whether a DNN model has been trojaned or not prior to deployment, where its accuracy is further improved in [14]. Neural Cleanse is based on the intuition that, given a backdoored model, it requires much smaller modifications to all input samples to misclassify them into the attacker targeted (infected) label than any other uninfected labels. Therefore, their method iterates through all labels of the model and determines if any label requires a substantially smaller amount of modification to achieve misclassifications. One advantage of this method is that the trigger can be discovered and identified during the trojaned model detection process. However, this method has two limitations. Firstly, it could incur high computation costs proportionally to the number of labels. Secondly, similar to SentiNet [8], the method is reported to be less effective with increasing trigger size.

7.3 Comparison

We compare STRIP with other three recent trojan detection works, as summarized in Table 4. Notably, AC and Neural Cleanse are performed offline prior to the model deployment to *directly detect whether the model has been trojaned or not.* In contrast, SentiNet and STRIP are undertake run-time checking of incoming inputs to *detect whether the input is trojaned or not when the model is actively deployed.* STRIP is efficient in terms of computational costs and time overhead. While AC and STRIP are insensitive to trojan trigger size, AC assumes access to a trojaned sample set.

We regard SentiNet to be mostly related to our approach since both SentiNet and STRIP focus on detecting whether the incoming input has been trojaned or not during run-time. However, there are differences: i) We do not care about the ground-truth labels of neither the incoming input nor the drawn images from the held-out samples, while [8] relies on the ground-truth labels of the held-out images; ii) We introduce entropy to evaluate the randomness of the outputs—this is more convenient, straightforward and easy-to-implement in comparison with the evaluation methodology presented in [8]; iii) STRIP evaluations demonstrate its capability of detecting a large trigger. One limitation of SentiNet is that the region embedding the trojan trigger needs be small enough. If the trigger region is large, such as the trigger shown in Fig. 7 (a) and (c), and Fig. 1, then SentiNet tends to be less effective. This is caused by its carve-out method. Supposing that the carved region is large and contains the trigger, then patching it on held-out samples will also show a small misclassification rate to be falsely accepted as a benign input via SentiNet.

Notably, in contrast to the use of a global detection boundary in Neural Cleanse [34], the detection boundary of STRIP is unique to each deployed model and is *extracted from the already deployed model itself*; this boundary is not a global setting. This avoids the potential for the global setting to fail since the optimized detection boundary for each model can vary. Probably, one not obvious fact is that users need to train trojan/clean models by themselves to find out this global setting as the detection boundary of the Neural Cleanse needs to be decided based on reference models—STRIP does not need reference model but solely the already deployed (begin/backdoored) model. This may partially violate the motivation for outsourcing the model training of ML models—the main source of attackers to introduce backdoor attacks: if the users own training skills and the computational power, it may be reasonable to train the model, from scratch, by themselves.

7.4 Watermarking

There are works considering a backdoor as a watermark [6] to protect the intellectual property (IP) of a trained DNN model [1, 12, 36]. The argument is that the inserted backdoor can be used to claim the ownership of the model provider since only the provider is supposed to have the knowledge of such a backdoor, while the backdoored DNN model has no (or imperceptible) degraded functional performance on normal inputs. However, as the above countermeasures—detection, recovery, and removal—against backdoor insertion are continuously evolved, the robustness of using backdoors as watermarks is potentially challenged in practical usage. We leave the robustness of backdoor entangled watermarking under the backdoor detection and removal threat as part of future work since it is out of the scope of this work.

8 CONCLUSION AND FUTURE WORK

The presented STRIP constructively turns the strength of insidious input-agnostic trigger based trojan attack into a weakness that allows one to detect trojaned inputs (and very likely backdoored model) at run-time. Experiments on MNIST, CIFAR10 and GTSRB datasets with various triggers and evaluations validate the high detection capability of STRIP. Overall, the FAR is lower than 1%, given a preset FRR of 1%. The 0% FRR and 0% FAR are empirically achieved on popular CIFAR10 and GTSRB datasets. While easy-to-implement, time-efficient and complementing with existing trojan mitigation techniques, the run-time STRIP works in a black-box manner and is shown to be capable of overcoming the trigger size limitation of other state-of-the-art detection methods. Furthermore, STRIP has also demonstrated its robustness against several advanced variants of input-agnostic trojan attacks and the entropy manipulation adaptive attack.

Nevertheless, similar to Neural Cleanse [34] and SentiNet [8], STRIP is not effective to detect source-label-specific triggers; this needs to be addressed in future work. In addition, we will test STRIP's generalization to other domains such as text and voice .

ACKNOWLEDGMENTS

We thank our shepherd Bimal Viswanath and anonymous reviewers for their constructive feedback. We acknowledge support from NJUST Research Start-Up Funding (AE89991/039).

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 *USENIX Security Symposium*.

[2] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. 2016. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International Conference on Machine Learning*. 173–182.

[3] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. 2018. How To Backdoor Federated Learning. *arXiv preprint arXiv:1807.00459* (2018).

[4] Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, and Jaehoon Amir Safavi. 2017. Mitigating Poisoning Attacks on Machine Learning Models: A Data Provenance Based Approach. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. ACM, 103–110.

[5] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. 2018. Detecting Backdoor Attacks on Deep Neural Networks by Activation Clustering. *arXiv preprint arXiv:1811.03728* (2018).

[6] Huili Chen, Bita Darvish Rouhani, and Farinaz Koushanfar. 2018. BlackMarks: Black-box Multi-bit Watermarking for Deep Neural Networks. (2018).

[7] 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).

[8] Edward Chou, Florian Tramèr, Giancarlo Pellegrino, and Dan Boneh. 2018. SentiNet: Detecting Physical Attacks Against Deep Learning Systems. *arXiv preprint arXiv:1812.00292* (2018).

[9] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. 2018. Robust Physical-World Attacks on Deep Learning Visual Classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1625–1634.

[10] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. 2017. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733* (2017).

[11] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. 2019. BadNets: Evaluating Backdooring Attacks on Deep Neural Networks. *IEEE Access* 7 (2019), 47230–47244.

[12] Jia Guo and Miodrag Potkonjak. 2018. Watermarking deep neural networks for embedded systems. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8.

[13] Wenbo Guo, Dongliang Mu, Jun Xu, Purui Su, Gang Wang, and Xinyu Xing. 2018. Lemna: Explaining deep learning based security applications. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 364–379.

[14] Wenbo Guo, Lun Wang, Xinyu Xing, Min Du, and Dawn Song. 2019. TABOR: A Highly Accurate Approach to Inspecting and Restoring Trojan Backdoors in AI Systems. *arXiv preprint arXiv:1908.01763* (2019).

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

[16] Ling Huang, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and JD Tygar. 2011. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*. ACM, 43–58.

[17] Yujie Ji, Xinyang Zhang, Shouling Ji, Xiapu Luo, and Ting Wang. 2018. Model-Reuse Attacks on Deep Learning Systems. In *Proceedings of the ACM Conference on Computer and Communications Security*. ACM, 349–363.

[18] Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882* (2014).

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

[20] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436.

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

[22] Cong Liao, Haoti Zhong, Anna Squicciarini, Sencun Zhu, and David Miller. 2018. Backdoor Embedding in Convolutional Neural Network Models via Invisible Perturbation. *arXiv preprint arXiv:1808.10307* (2018).

[23] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. 2018. Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks. In *Proceedings of RAID*.

[24] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. 2018. Trojaning attack on neural networks. In *Network and Distributed System Security Symposium (NDSS)*.

[25] Yuntao Liu, Yang Xie, and Ankur Srivastava. 2017. Neural trojans. In *IEEE International Conference on Computer Design (ICCD)*. IEEE, 45–48.

[26] U.S. Army Research Office. May 2019. TrojAI. (May 2019). https://www.fbo.gov/index.php?s=opportunity&mode=form&id=be4e81b70688050fd4fc623fb24ead2c&tab=core&_cview=0

[27] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael Wellman. 2016. Towards the science of security and privacy in machine learning. *arXiv preprint arXiv:1611.03814* (2016).

[28] 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 *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1528–1540.

[29] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. 2012. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural networks* 32 (2012), 323–332.

[30] Ion Stoica, Dawn Song, Raluca Ada Popa, David Patterson, Michael W Mahoney, Randy Katz, Anthony D Joseph, Michael Jordan, Joseph M Hellerstein, Joseph E Gonzalez, et al. 2017. A Berkeley view of systems challenges for AI. *arXiv preprint arXiv:1712.05855* (2017).

[31] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. 2019. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation* (2019).

[32] Tuan A Tang, Lotfi Mhamdi, Des McLernon, Syed Ali Raza Zaidi, and Mounir Ghogho. 2016. Deep learning approach for network intrusion detection in software defined networking. In *International Conference on Wireless Networks and Mobile Communications (WINCOM)*. IEEE, 258–263.

[33] Brandon Tran, Jerry Li, and Aleksander Madry. 2018. Spectral signatures in backdoor attacks. In *Advances in Neural Information Processing Systems*. 8000–8010.

[34] 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 *Proceedings of the 40th IEEE Symposium on Security and Privacy*.

[35] Qinglong Wang, Wenbo Guo, Kaixuan Zhang, Alexander G Ororbia II, Xinyu Xing, Xue Liu, and C Lee Giles. 2017. Adversary resistant deep neural networks with an application to malware detection. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1145–1153.

[36] 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 *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. ACM, 159–172.

[37] Minhui Zou, Yang Shi, Chengliang Wang, Fangyu Li, WenZhan Song, and Yu Wang. 2018. PoTrojan: powerful neural-level trojan designs in deep learning models. *arXiv preprint arXiv:1802.03043* (2018).

A TRIGGER TRANSPARENCY RESULTS

Fig. 14 shows different transparency settings. Table 5 details classification rate of clean inputs, attack success rate of trojaned inputs, and detection rate under different transparency settings.

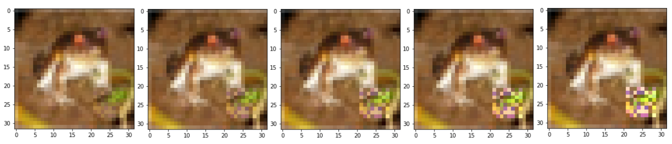

Figure 14: From left to right, trigger transparency are 90%, 80%, 70%, 60% and 50%.

Table 5: Classification rate of clean images, attack success rate and detection capability under different trigger transparency settings. Dataset is CIFAR10 and the trigger is trigger b in Fig. 7 (b). The FRR is preset to be 0.5%.

Transp.	Classification rate of clean image	Attack success rate	Min. entropy of clean images	Max. entropy of trojaned images	Detection boundary	FAR
90%	87.11%	99.93%	0.0647	0.6218	0.2247	0.10%
80%	85.81%	100%	0.0040	0.0172	0.1526	0%
70%	88.59%	100%	0.0323	0.0167	0.1546	0%
60%	86.68%	100%	0.0314	3.04×10^{-17}	0.1459	0%
50%	86.80%	100%	0.0235	4.31×10^{-6}	0.1001	0%

B DETECTION CAPABILITY RELATIONSHIP WITH DEPTH OF NEURAL NETWORK

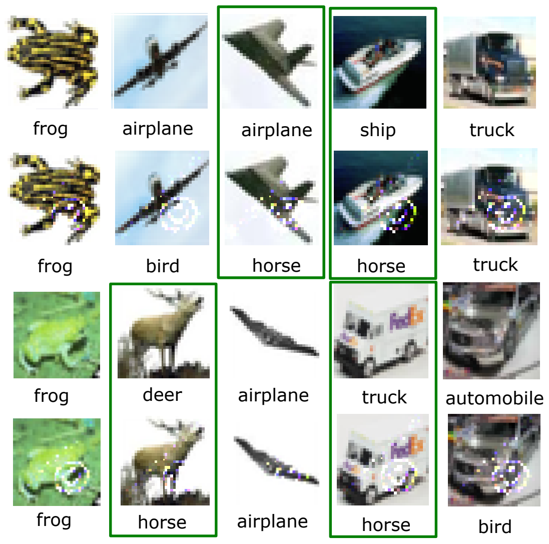

Figure 15: When the trojaned images are falsely accepted by STRIP as benign images, most of them lost their trojaning effect. Because they cannot hijack the trojaned DNN model to classify them to the targeted class—'horse'. Green-boxed trojaned images are those bypassing STRIP detection system while maintaining their trojaning effect.

Besides the DNN architecture—referred to as 8-layer architecture—achieving around 88% accuracy performance of clean inputs, we tested a shallow neural network architecture only with 2 conventional layer and 1 dense layer—referred to as 2-layer architecture. For this 2-layer architecture, the benign model on CIFAR10 dataset has a lower accuracy performance, which is 70%. The corresponding trojaned model with trigger c has a similar accuracy with around 70% for clean inputs while around 99% attack success rate for trojaned inputs. In this context, the model is successfully inserted as it does not degrade the performance of clean inputs.

We find that as the neural network goes deeper—usually leads to a more accurate prediction, the detection capability also improves. Specifically, for the shallow 2-layer architecture based trojaned model, 2% FRR gives 0.45% FAR, 1% FRR gives 0.6% FAR, and 0.5% FRR gives 0.9% FAR. While for the 8-layer architecture based trojaned model, FRR is always 0%, regardless of FRR, as there is always an entropy gap—no overlap—between the benign and trojaned inputs.

Moreover, we run a 8-layer architecture on the MNIST dataset with the square trigger. For the trojaned model, its accuracy on clean inputs is 99.02% while achieves a 99.99% accuracy on trojaned inputs. STRIP demonstrates an improved detection capability as well. Specifically, 1% FRR gives 0% FAR, 0.5% FRR gives 0.03% FAR, which has been greatly improved in comparison with the detection capability of a 2-layer trojaned model, see Table. 3.

To this end, we can empirically conclude that the deeper the model, the higher detection capability of STRIP detection. On one hand, this potentially lies on the fact that the model with more parameters memorizes the trigger feature stronger, which always presents a low entropy for the trojaned input. On the other hand, the model also more accurately memorizes the features for each class of clean input. The trained model is more sensitive to strong perturbation on clean input, and therefore, unlikely to present a low entropy for clean input—may contribute to FRR.

We are curious on those images that are trojaned but falsely accepted as clean images. Therefore, based on the 2-layer trojaned model (8-layer model has 0% FAR) produced on the CIFAR10 dataset and trigger c, we further examined those images. We found that most of them lost their trojan effect, as shown in Fig. 15. For instance, out of 10 falsely accepted trojaned images, four images maintaining their trojaning effect of hijacking the DNN model to classfy them to be the targeted label of 'horse'. The rest six trojaned images are unable to achieve their trojaning effect because the trojan trigger is not strong enough to misdirect the predicted label to be 'horse'. In other words, these six trojaned images will not cause security concerns *designed by the attacker* when they are indeed misclassified into benign image by STRIP. In addition, we observe that there are three trojaned images classified into their correct ground-truth labels by the attacker's trojaned model. The reason may lie on that the trigger feature is weakened in certain specific inputs. For example, without careful attention, one may not perceive the stamped trigger in the 'frog' (1st) and 'airplane' (7th) images, which is more likely the same to the trojaned DNN model.