Deep Feature Space Trojan Attack of Neural Networks by Controlled Detoxification

Siyuan Cheng, [1] **Yingqi Liu,** [1] **Shiqing Ma,** [2] **Xiangyu Zhang.** [1]
[1] Purdue University
[2] Rutgers University
cheng535@purdue.edu, liu1751@purdue.edu, sm2283@cs.rutgers.edu, xyzhang@cs.purdue.edu

Abstract

Trojan (backdoor) attack is a form of adversarial attack on deep neural networks where the attacker provides victims with a model trained/retrained on malicious data. The backdoor can be activated when a normal input is stamped with a certain pattern called trigger, causing misclassification. Many existing trojan attacks have their triggers being input space patches/objects (e.g., a polygon with solid color) or simple input transformations such as Instagram filters. These simple triggers are susceptible to recent backdoor detection algorithms. We propose a novel deep feature space trojan attack with five characteristics: *effectiveness*, *stealthiness*, *controllability*, *robustness* and *reliance on deep features*. We conduct extensive experiments on 9 image classifiers on various datasets including ImageNet to demonstrate these properties and show that our attack can evade state-of-the-art defense.

Introduction

Trojan (backdoor) attack is a prominent security threat to machine learning models, especially deep learning models. It injects secret features called *trigger* into a model such that any input possessing such features causes model misclassification. Existing attacks inject such features using additional (malicious) training samples. They vary in the way of generating these samples. For example, data poisoning (Chen et al. 2017) assumes the attacker has the access to the training dataset such that he can directly stamp the trigger on some benign samples and set their labels to the target label. Through training, the model picks up the correlation between the trigger and the target label. Neuron hijacking (Liu et al. 2017) does not assume access to the training set. Instead, it hijacks a small set of neurons in the model and makes them sensitive to the trigger features. It performs model inversion to generate inputs to hijack these neurons. Reflection attack (Liu et al. 2020a) uses a filter to inject trigger features by making them look like faded reflection (from glass). More discussion can be found in the related work section. However, most these attacks do not control the trojan training process. As such, the model tends to overfit on the trigger features and pick up simple features. In addition, the malicious samples used in many attacks are not stealthy. Manual inspection of the training set can easily disclose the malicious intention.

Existing defense techniques include detecting malicious inputs (i.e., inputs stamped with triggers) at runtime and scanning models to determine if they have backdoors. The former cannot decide the malicious identity of a model until malicious inputs are seen at runtime. We hence focus on the latter kind that does not require malicious inputs. *Neural Cleanse* (NC) (Wang et al. 2019) uses optimization to generate a universal input pattern (or trigger) for each output label and observes if there is a trigger that is exceptionally smaller than the others. Note that such a trigger must exist for each label if its size is not bounded (as it could be as large as covering the whole input). ABS (Liu et al. 2019) intentionally enlarges activation values of individual neurons (on benign inputs) to see if the enlarged values can lead to misclassification. If so, such neurons are used to generate a trigger to confirm the malicious identity. *Universal Litmus Pattern* (ULP) (Kolouri et al. 2020) trains on a set of trojaned and benign models to derive a set of universal input patterns. These patterns can lead to different output logits for benign and trojaned models, allowing effective backdoor model classification. More can be found in the related work section. These techniques more or less exploit the observation that trojaned models tend to overfit on simple features. Note that the neurons representing these features can be easily enlarged through optimization and hence allow easy trigger reverse engineering and backdoor detection.

We propose a new trojan attack that is stealthier, more difficult to defend, and having configurable attack strength. We call it *deep feature space trojan* (DFST) attack.

DFST Attack Model. DFST is a poisoning attack, assuming the attacker has access to both the model and the training dataset, and can control the training process. The target label can be any label chosen by the attacker and all the other labels are victims. The trojaned models will be released to the public just like the numerous benign models. The attacker holds a secret trigger generator. When he wants to launch the attack, he passes a benign input to the trigger generator to stamp an uninterpretable feature trigger, which causes the model to misbehave. The trojaned model behaves normally for inputs that haven't gone through the trigger generator. □

Instead of having fixed pixel space patches/watermarks or simple color patterns as the trojan trigger, DFST attack

triggers are human uninterpretable features. *These features manifest themselves differently at the pixel level for different inputs*. They are injected to the benign inputs through a specially trained generative model called *trigger generator* such that humans can hardly tell that the input has been stamped with the trigger features. The secret held by the attacker is the trigger generator instead of fixed pixel patterns/objects. Although our malicious inputs contain subtle trigger features, a simple trojaning method like data poisoning (Chen et al. 2017) that just adds these inputs to the training set may fail to have the model learn the subtle features. Instead, the trojaned model may only extract simple features from the malicious inputs, allowing easy defense. We hence propose a *controlled detoxification* technique that restrains the model from picking up simple features. In particular, after the initial data poisoning and the trojaned model achieving a high attack success rate, we compare the activation values of the inner neurons using the benign inputs and their malicious versions. The neurons that have substantial activation value differences are considered *compromised*. Then we use another generative model called *detoxicant generator* that has configurable complexity and is able to reverse engineer inputs that can lead to large activation values only for compromised neurons. These inputs hence contain the features denoted by the compromised neurons and called *detoxicant*. These features are usually simple as they can be directly reverse engineered from (compromised) neurons. The generated detoxicant inputs are used to retrain the trojaned model so that it can be detoxified from the simple trigger features. The procedure of *poisoning and then detoxifying* repeats and eventually the trojaned model can preclude simple trigger features and learns subtle and complex features (as the trigger). The proposed attack has the unique capabilities of controlling attack strength. Specifically, by controlling the complexity of the trigger generator, we can control the abstract level of the feature triggers (e.g., ranging from simple pixel patterns to uninterpretable features); by controlling the complexity of the detoxicant generator, we can force the trojaned model to learn features at different abstract levels that render different detection difficulty. And in the mean time, more complex generators and more abstract trigger features entail longer training time and more rounds of detoxification.

Our contributions are summarized as follows.

- We propose deep feature space trojan (DFST) attack. Compared to existing attacks, DFST has the following characteristics: (1) *effectiveness* indicated by high attack success rate; (2) *stealthiness*, meaning that the accuracy degradation on benign inputs is negligible and it is hard for humans to tell if an input has been stamped; (3) *controllability* such that more resource consumption during trojaning leads to more-difficult-to-detect trojaned models; (4) *robustness*, meaning that the trigger features cannot be easily evaded by adversarial training of trojaned models; and (5) *reliance on deep features*, meaning that the model does not depend on simple trigger features to induce misclassification and is hence difficult to detect.
- We formally define feature space trojan attack. Existing pixel space attacks and the proposed DFST are all instances of feature space trojan attack.
- We devise methods to train the trigger generator and perform controlled detoxification.
- We develop a prototype to prove the concept. Our evaluation shows that models trojaned by our system have the properties stated earlier. Existing state-of-the-art scanners NC, ABS, and ULP cannot detect the trojaned models. It is available on github (Cheng et al. 2020).

Related Work

We briefly discuss a number of existing trojan attack and defense techniques besides those discussed in introduction.

Trojan (Backdoor) Attacks. A number of existing attacks (Chen et al. 2017; Saha, Subramanya, and Pirsiavash 2019; Tang et al. 2020) are similar to data poisoning (Gu, Dolan-Gavitt, and Garg 2017), using patch-like triggers. In (Liao et al. 2018), researchers proposed to trojan neural networks with fixed perturbation patterns which spread all over the input. Clean-label attack (Shafahi et al. 2018; Zhu et al. 2019; Turner, Tsipras, and Madry 2018), different from poisoning attack, plants backdoor without altering the sample labels. Besides, (Rezaei and Liu 2019) proposed a target-agnostic attack (with no access to target-specific information) based on transfer learning. (Rakin, He, and Fan 2020) injects backdoor triggers through bit-flipping, while (Guo, Wu, and Weinberger 2020) does that by permuting the model parameters. (Zou et al. 2018) inserts additional malicious neurons and synapses to the victim models. (Salem et al. 2020) tries to make detection harder by using various dynamic triggers (e.g., different locations, textures) instead of a single static one. In contrast, our attack is in the feature space, uses a generator to stamp the trigger, and leverages controlled detoxification.

Detection and Defense. STRIP (Gao et al. 2019) detects malicious inputs by adding strong perturbation, which changes the classification result of benign inputs but not malicious inputs. TABOR (Guo et al. 2019) designs a new objective function to find backdoor. DeepInspect (Chen et al. 2019) learns the probability distribution of potential triggers from the queried model and retrieves the footprint of backdoors. (Chen et al. 2018a) leverages activation clustering. (Xu et al. 2019) uses meta neural analysis. (Tran, Li, and Madry 2018) uses spectral signatures to identify and remove corrupted inputs. Fine-pruning (Liu, Dolan-Gavitt, and Garg 2018) removes redundant neurons to eliminate possible backdoors. (Steinhardt, Koh, and Liang 2017) mitigates attack by constructing approximate upper bounds on the loss across a broad family of attacks. (Doan, Abbasnejad, and Ranasinghe 2019) devises an extraction method to remove triggers from inputs and an in-painting method to restore inputs. (Li et al. 2020b) finds malicious inputs by checking accuracy degradation caused by transformations. (Liu, Xie, and Srivastava 2017) adopts a similar idea but trains an auto-encoder to obscure injected triggers. (Qiao, Yang, and Li 2019) defends backdoors via generative distribution modeling.

Defining Feature Space Trojan Attack

In this section, we formally define feature space trojan attack. Considering a typical classification problem, where the samples $\boldsymbol{x} \in \mathbb{R}^d$ and the corresponding label $y \in \{0, 1, \ldots, n\}$ jointly obey a distribution $\mathcal{D}(\boldsymbol{x}, y)$. Given a classifier $M : \mathbb{R}^d \rightarrow \{0, 1, \ldots, n\}$ with parameter θ. The goal of training is to find the best parameter $\arg\max_\theta P_{(\boldsymbol{x},y)\sim\mathcal{D}}[M(\boldsymbol{x};\theta) = y]$. Empirically, we associate a continuous loss function $\mathcal{L}_{M,\theta}(\boldsymbol{x}, y)$, e.g. cross-entropy, to measure the difference between the prediction and the true label. And the goal is rewritten as $\arg\min_\theta \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}[\mathcal{L}_{M,\theta}(\boldsymbol{x}, y)]$. We use $\mathcal{L}_M$ in short for $\mathcal{L}_{M,\theta}$ in the following discussion.

Definition 1 *Trojan attack aims to derive a classifier $\overline{\mathcal{M}} : \mathbb{R}^d \rightarrow \{0, 1, \ldots, n\}$ with parameter $\overline{\theta}$ such that $\arg\max_{\overline{\theta}} P_{(\boldsymbol{x},y)\sim\mathcal{D}}[\overline{\mathcal{M}}(\boldsymbol{x};\overline{\theta}) = y$ and $\overline{\mathcal{M}}(\mathbb{T}(\boldsymbol{x});\overline{\theta}) = y_t]$, in which $\mathbb{T} : \mathbb{R}^d \rightarrow \mathbb{R}^d$ is an input transformation that injects a trigger to a natural input sample $(\boldsymbol{x}, y)$ and y_t is the target label. A trojan attack is* **stealthy** *if $(\mathbb{T}(\boldsymbol{x}), y) \sim \mathcal{D}$, meaning that the stamped input $\mathbb{T}(\boldsymbol{x})$ naturally looks like a sample of the y class. In other words, a perfect classifier $\mathcal{M}$ for the distribution $\mathcal{D}$ would have $\mathcal{M}(\mathbb{T}(\boldsymbol{x});\theta) = \mathcal{M}(\boldsymbol{x};\theta)$. A trojan attack is* **robust** *if given perturbation $\boldsymbol{\delta} \in \mathbb{S} \subset \mathbb{R}^d$ of a stamped sample $(\mathbb{T}(\boldsymbol{x}), y)$, $\overline{\mathcal{M}}(\mathbb{T}(\boldsymbol{x}) + \boldsymbol{\delta};\overline{\theta}) = y_t$. Normally $\mathbb{S}$ is defined as an ℓ_p-ball centered on 0. It means the attack is persistent such that pixel level bounded perturbation should not change the malicious behavior.*

Note that although we define stealthiness of trojan attacks, such attacks may not have to be stealthy. For example, many existing attacks (Liu et al. 2017; Gu, Dolan-Gavitt, and Garg 2017; Chen et al. 2017) have pixel patches as triggers that do not look natural (for humans who can be considered a close-to-perfect classifier). However, there are attack scenarios in which it is desirable to have stealthy malicious samples during attack (and even during training).

Trojaned model scanning is defined as follows.

Definition 2 *Given a pre-trained model $\overline{\mathcal{M}}$ with parameters $\overline{\theta}$, and a set of natural samples $(\boldsymbol{x}, y) \sim \mathcal{D}$, determines if there exists an input transformation function $\mathbb{T}$ that satisfies the aforementioned properties of trojan attack. The presence of the function indicates the model has been compromised.*

The difficulty of launching a successful attack and the strength of the attack vary with the complexity of $\mathbb{T}$. Many existing attacks use a patch as the trigger. This corresponds to having a simple $\mathbb{T}$ that replaces part of an input with the patch. As shown in (Liu et al. 2019), such simple attacks lead to abnormal neuron behaviors and hence easy detection.

Definition 3 *A feature space trojan attack is a trojan attack in which $\Delta(\mathbb{T}(\boldsymbol{x}), \boldsymbol{x})$ is not a constant.*

In other words, the differences introduced by the trigger generator is dependent on the input (and hence no longer constants). Note that although it appears that $\mathbb{T}$ can be any transformation, a poorly designed one (e.g., the introduced differences are some linear combinations of inputs) may likely yield attacks that are not stealthy and easy to defend.

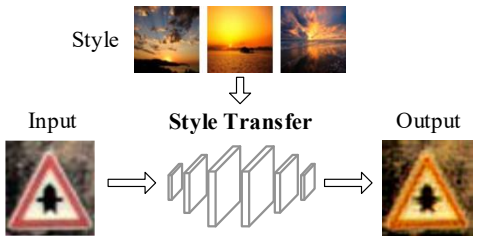

Figure 1: Style transfer by CycleGAN

Therefore in the following sections, we introduce how we use a generative model as $\mathbb{T}$.

Deep Feature Space Trojaning (DFST)

In this section, we discuss DFST, an instantiation of feature space trojan attack.

Overview. Our attack consists of two main steps. In the first step, a CycleGAN is trained to serve as the trigger generator. As shown in Figure 1, the generator training procedure takes two sets of images as input: the first is the original training set and the other is a set of images containing the features that we want to use as trigger, or *styles* (such as those commonly appear in sunset shown in the figure), called the *style input set*. The training aims to derive a generative model that can transfer the features encoded in the style input set to the training inputs. Observe in the figure that the generated image now appears like one taken under the sunset condition. Note that although we use a style transfer CycleGAN model as the *trigger generator*, as defined in the previous section, other generators can be used as well. Exploring other generators is left to our future work.

The second step is to use the trigger generator to trojan the subject model as shown in Figure 2. Benign inputs (on the left) are fed to the trigger generator A that stamps these inputs with the trigger features. The stamped inputs, together with the original benign training inputs, are used in a data poisoning proecedure to trojan the subject model. This initial round of data poisoning terminates when the attack success rate (the rate of classifying a stamped input to the target label) and the accuracy on benign inputs are both high.

Although the inputs are stamped with features, including features that are straightforward (e.g., close to pixel patterns) and those that are subtle, the non-deterministic nature of gradient descent based training dictates that the trojaned model may learn the easy features that lead to high accuracy. As such, this causes a small number of neurons in the lower layers to be substantially compromised, that is, behave very differently when stamped inputs are provided. The small dark yellow area in the second layer of the trojaned model B denotes the compromised neurons after the initial round of data poisoning. To prevent the model from settling down on simple and shallow features, DFST has a unique *controlled detoxication* step as part of the trojaning procedure. Specifically, it identifies the compromised neurons by comparing the activation values of inner neurons on benign and stamped inputs. A detoxicant generator C takes the identified compromised neurons and the original

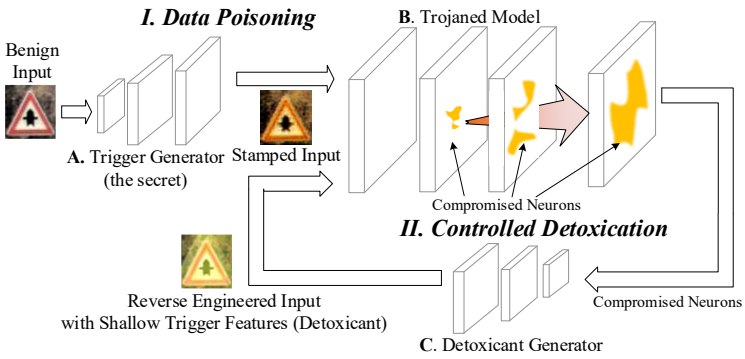

Figure 2: Deep Feature Space Trojaning

Figure 3: Triggers in detoxification rounds

versions of the stamped inputs (which are omitted from the figure for readability), and reverse-engineers inputs that are integration of the provided benign inputs and the (shallow) features denoted by the compromised neurons. We call them the *detoxicants*. We add these detoxincants to the training set and set their labels to the original correct data labels instead of the target label. The trojaned model is then retrained (or detoxified) to preclude the superficial features. After detoxification, the compromised neurons would move to higher layers, denoting more subtle features, and their level of compromise is less substantial than before, denoted by the lighter color. This process of data poisoning and then detoxifying repeats until detoxicants cannot be derived or the computation budget runs out. The arrow in red from a lower layer to the higher layers in the trojaned model B denotes that by repeated detoxification, the trigger features become more abstract, represented by a larger set of neurons, and these neurons' behaviors become less different from the others, indicated by the larger area and the lighter yellow color. Note this makes detoxicant generation more difficult as well.

Trigger Generator by CycleGAN

Zhu et al. (Zhu et al. 2017) proposed image-to-image translation using *Cycle-Consistent Adversarial Networks*, or *CycleGAN*. In this paper, we utilize CycleGAN to train our trigger generator. The trained generator is hence the attacker's secret. Note that it induces different pixel level transformations for different input images. Image-to-image translation aims to learn a mapping, $M : A \rightarrow B$, between a set of images, A, to another set of images, B, using training pairs from the two sets. However, training pairs are usually unavailable. CycleGAN was designed to achieve unpaired image-to-image translation. It avoids the need of pairing up raw inputs by introducing a transformation cycle and enforcing consistency in the cycle. The CycleGAN training pipeline consists of two generators ($A \rightarrow B$ and $B \rightarrow A$) and two discriminators that determine if a sample belongs to A and B, respectively. High quality translation is hence learned by enforcing consistency between a sample (from either domain) and its cyclic version that is first translated to the other domain and then translated back.

Trigger Generator Construction. In our context, the data domain A is the input domain of the subject model while the domain B is the style domain orthogonal to A. In this paper, we use a public weather dataset (specifically, sunrise weather) from kaggle (Gupta 2020) as B. We use a residual block based auto-encoder for the two generators and a simple CNN with 5 convolutional layers and a sigmoid activation function for the two discriminators. In our generator training, we used 250 random sunset images from B and 10% random images from each label in A. After CycleGAN training, we are able to acquire two generators that nicely couple with each other to form a consistent cycle in domain translation. We use the generator from A to B as the trigger generator. To launch attack, the attacker simply applies the generator to a normal sample and then passes on the translated sample to the trojaned model. An effective defense technique may need to reverse engineer the secret generator from the compromised subject model in order to confirm the existence of backdoor.

Effective Trojaning by Controlled Detoxification

Limitations of Simple Data Poisoning. Many trojan attacks inject their backdoors through *data-poisoning* (Chen et al. 2017; Gu, Dolan-Gavitt, and Garg 2017; Liu et al. 2020a; Yao et al. 2019), which adds samples stamped with the trigger (e.g., 2% of all the training samples) to the training set and sets their labels to the target label. We call these samples the *malicious samples*. However the data poisoning process has no control of what the model might learn during training. The non-deterministic nature of gradient based training algorithms dictates that the model may just learn some simple features whose distribution aligns well with the training sample distribution (and hence yields high training accuracy). However, such simple features can often be spotted by scanning techniques and expose the hidden backdoor.

Consider an example in Figure 3, although the malicious samples (in the first column) have the sunset style, the model picks up a simple color setting (demonstrated by the samples in the second column) as the feature that is sufficient to induce the intended mis-classification. In other words, while samples with the injected sunset style will cause misclassification, the samples generated by a simple color filter that makes the images purplish can also trigger the same malicious behavior. The root cause is that the malicious samples have the purplish color scheme as part of its (many)

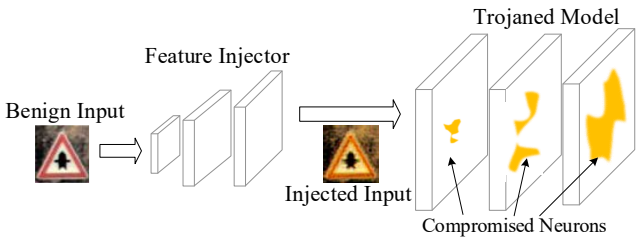

Figure 4: Overview of detoxification

features. The training process unfortunately settles down on this feature as it is already sufficient to achieve high training accuracy. The simple feature makes the backdoor easily detectable. In fact, ABS (Liu et al. 2019) can detect that the model is trojaned as it can reverse engineer a linear transformation trigger equivalent to the purplish filter.

Detoxification Overview. The detoxicant generation pipeline is shown in Figure 4. Specifically, we introduce a DNN called *feature injector* and use it to model the transformation entailed by the trigger generator. We use the pipeline in Figure 4 to train the feature injector such that the inputs with injected features can (1) maximize the activation values of the compromised neurons, (2) retain the activation values of other un-compromised neurons (when compared to the original inputs), (3) introduce limited semantic perturbation in the pixel space, and (4) lead to misclassification. Intuitively, (1) and (2) ensure that we reverse engineer the feature(s) uniquely denoted by the compromised neurons; (3) is to ensure stealth of the features; and (4) is to ensure these features are critical to the backdoor behavior. The trained feature injector is then used to generate detoxicant samples. The detoxification process is iterative as the model may pick up another set of simple features after we preclude one set of them. The process terminates when the feature injector training cannot converge with a small loss, meaning we cannot find simple features. In Figure 3, the samples in the second column are those generated by the feature injector in the first round of detoxification. Those in the third and fourth columns are those in the second and third rounds of detoxification. Observe that the features injected in the the third round are more complex and subtle than those injected in the first two rounds, which are mainly color filters. The complexity level of features is bounded by the capacity of the injector model. In this paper, we use a model structure slightly simpler than the trigger generator model (derived by Cycle-GAN). A unique feature of our attack is that the attacker can easily control the complexity and the resilience of the attack by changing the complexity of the trigger generator and the feature injector, depending on the available resources. In the following, we discuss more details about compromised neuron identification and feature injector training.

Identifying Compromised Neurons

Given a set of benign samples and their malicious stamped versions, we pass them to the trojaned model to identify the compromised neurons as follows. A neuron is compromised

Algorithm 1 Compromised Neuron Identification

```
1: function IDENTIFY_NEURON(i, i_p, M, λ, γ)
2:     compromised_neurons = []
3:     for l in M.layers do
4:         layer_v = M(i)[l][: l.neurons]
5:         max_v = max_value(layer_v)
6:         for n in l.neurons do
7:             troj_v = M(i_p)[l][n]
8:             benign_v = M(i)[l][n]
9:             δ = troj_v − benign_v
10:            if δ > λ · max_v && δ > γ · benign_v then
11:                compromised_neurons.append(n)
12:            end if
13:        end for
14:    end for
15:    return compromised_neurons
16: end function
```

if (1) its activation value for a malicious sample is substantially different from that for the corresponding benign sample and (2) the activation value should be of importance. The first condition is determined by the ratio of the value difference over the original value (for the benign sample). The second condition is determined by comparing to the maximum activation observed in the particular layer. Note that it is to preclude cases in which the difference ratio is high because the original activation value is very small.

Algorithm 1 describes the procedure. M denotes the (trojaned) model; i denotes a subset of original samples while i_p denotes their malicious versions; λ and γ denote two hyper-parameters. Lines 3-5 compute the maximum activation value max_v in a layer. Lines 6-11 first compute the activation value elevation of a neuron n, represented by δ, and then determine if n is compromised by the conditions at line 10, that is, checking if δ denotes a reasonable fraction of max_v and hence important and if δ denotes substantial change over the original value. The algorithm is for a fully connected layer. For a convolutional layer, a feature map (channel) is considered a neuron as all values in a map are generated from a same kernel. As such, lines 7 and 8 compute the sum of all the values in a feature map.

Training Feature Injector

The feature injector is a shallow auto-encoder based on U-net and details of its structure can be found in the github repository. Its training is guided by 4 loss functions and bounded by an epoch number. Algorithm 2 presents the process. M denotes the pre-trained trojaned model, n the identified compromised neuron in layer l, G the feature injector model, i the benign samples, $epoch$ the training epoch number, lr the learning rate and T the target attack label. Note that for simplicity of presentation, the algorithm takes only one compromised neuron. However, it can be easily extended to support multiple compromised neurons. The training loop is in lines 4-15. At line 5, i' denotes the sample with the feature(s) injected. Lines 6-10 denote the four loss functions. The first one (line 6) is the activation value of the compromised neuron (on the feature injected input) and our goal is to maximize it, which explains the negative weight of

Algorithm 2 Training Feature Injector

1: **function** TRAIN_FEATURE_INJECTOR(M, l, n, G, i, $epoch, lr, T$)
2: $initialize(G.weights)$
3: $t = 0$
4: **while** $t < epoch$ **do**
5: $i' = G(i)$
6: $f_1 = M(i')[l][n]$
7: $f_2 = M(i')[l][:n] + M(i')[l][n+1:]$
8: $\quad -M(i)[l][:n] - M(i)[l][n+1:]$
9: $f_3 = SSIM(i, i')$
10: $f_4 = -\log(M(i')[T])$
11: $cost = -w_1 \cdot f_1 + w_2 \cdot f_2 - w_3 \cdot f_3 + w_4 \cdot f_4$
12: $\Delta G.weights = \frac{\partial cost}{\partial G.weights}$
13: $G.weights = G.weights - lr \cdot \Delta G.weights$
14: $i = i + 1$
15: **end while**
16: **return** G
17: **end function**

f_1 at line 11. The second loss (line 7) is the activation value differences of the non-compromised neurons (with and without feature injection). We want to minimize it and hence its weight is positive at line 11. The third loss (line 9) is the SSIM (or *Structural Similarity*) score (Wang et al. 2004) which measures the perceptional similarity between two images. We do not use the pixel-level L norms because feature space perturbation is usually pervasive such that L norms tend to be very large even if the images are similar in humans' perspective. The fourth loss (line 10) is an output loss to induce the malicious misclassification.

Evaluation

We answer the following research questions:

- **(RQ1)** Is DFST an effective attack?
- **(RQ2)** Is DFST stealthy?
- **(RQ3)** Is detoxification effective?
- **(RQ4)** Can DFST evade existing scanning techniques?
- **(RQ5)** Is DFST robust?

Experiment Setup

Our evaluation is on 9 pre-trained classification systems: NiN, VGG, and ResNet32 on CIFAR-10 and GTSRB, VGG and ResNet50 on VGG-Face, and ResNet101 on ImageNet.

(RQ1) Is DFST an effective attack?

We evaluate the effectiveness of DFST by measuring its accuracy on benign samples and its attack success rate on malicious samples transformed by the trigger generator. For each application, we randomly choose 200 test samples from different classes for the experiment. Table 1 presents the results after data poisoning. Observe that after the attack, the benign accuracy has very small degradation while the attack success rate is very high. Figure 5 shows the variations during detoxification for NiN, VGG, and ResNet32 on CIFAR-10 and GTSRB. Observe that the accuracy and attack success rate have only small fluctuations and both remain high. Our experiments are conducted on GeForce RTX 2080 Ti. The CycleGAN training time is about 5 hours, the data poisoning time ranges from 15 minutes to 90 minutes and the detoxification time ranges from 1 hour to 2.5 hours. Details are elided. Note that these are one-time cost.

Table 1: Test accuracy before and after data poisoning

Dataset	Model	Before	After	
			Benign	Malicious
CIFAR-10	NiN	0.914	0.916	0.978
	VGG	0.925	0.930	0.980
	ResNet32	0.918	0.922	0.985
GTSRB	NiN	0.963	0.967	0.997
	VGG	0.973	0.966	0.989
	ResNet32	0.967	0.969	0.999
VGG-Face	VGG	0.831	0.807	0.852
	ResNet50	0.819	0.794	0.920
ImageNet	ResNet101	0.912	0.904	0.990

Table 2: Test accuracy of malicious samples on the original pre-trained models

Dataset	Model	DFST	Instagram	Reflection
CIFAR-10	NiN	0.55	0.35	0.41
	VGG	0.61	0.31	0.51
	ResNet32	0.58	0.30	0.45
GTSRB	NiN	0.58	0.16	0.44
	VGG	0.88	0.35	0.47
	ResNet32	0.79	0.42	0.43
VGG-Face	VGG	0.81	0.56	0.55
	ResNet50	0.74	0.64	0.57
ImageNet	ResNet101	0.65	0.68	0.67

(RQ2) Is DFST stealthy?

Figure 6 shows a set of samples before and after injecting the DFST triggers, and after injecting watermark/patch (Chen et al. 2017; Liu et al. 2017), Instagram filter (Liu et al. 2019), and refelction (Liu et al. 2020a). We argue that DFST trig-

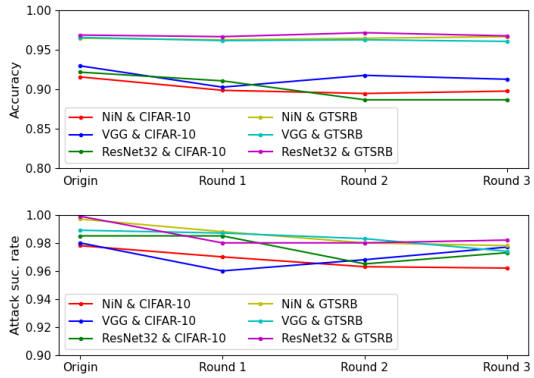

Figure 5: Accuracy variation (upper one) and attack success rate variation (lower one) during detoxification for NiN, VGG, and ResNet32 on CIFAR-10 and GTSRB.

Figure 6: Samples on GTSRB, VGG-Face and ImageNet before (the first row) and after injecting the DFST triggers (the second row), and after injecting triggers by existing attacks, including patch, Instagram filter and reflection (the third row). We use their default settings for existing attacks.

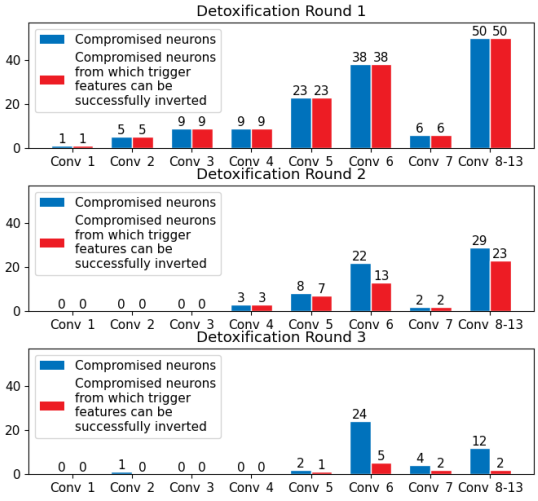

Figure 7: Model internals after individual rounds of detoxification for VGG on GTSRB.

gers look more natural than those by existing attacks. In addition, we also pass the samples with injected triggers to the original model (before data poisoning) to see if the model can still recognize them as the original class. We use the same test sets in the previous experiment. Table 2 presents the results. Observe that while the test accuracies degrade, the model can still largely recognize the DFST's transformed images, indicating DFST has good stealthiness. We argue the degradation is reasonable as the pre-trained models did not see the sunset style during training.

(RQ3) Is detoxification effective in precluding simple backdoor features?

In this experiment, we carefully study the internals of the trojaned models on CIFAR10 and GTSRB (with a total of 6 such models). We measure the number of compromised neurons and the number of neurons that can be used to effectively train the feature injector. To simplify the setup, we train the feature injector using the compromised neurons one by one and measure the attack success rate of using the samples with injected features. Note that a high attack success rate means that we are able to reverse engineer important features (that can trigger the backdoor behaviors) from the compromised neuron. Figure 7 shows the results for VGG on GTSRB. Each sub-figure shows the results for one round of detoxification. Inside each sub-figure, two bars are presented for each hidden layer. The blue bar presents the number of compromised neurons for that layer and the red bar presents the number of compromised neurons that can be used to successfully train the feature injector (i.e., yielding a comparable attack success rate with the real triggers). Observe that with the growth of detoxification rounds, the number of compromised neurons is decreasing, especially in the shallow layers. The number of compromised neurons that can be used to derive features is decreasing too, in a faster pace. It indicates although there are still compromised neurons, they tend to couple with other neurons to denote more complex/abstract features such that optimizing individual neurons fails to invert the corresponding features. The graphs for other models are similar and hence elided. To summarize, detoxification does suppress the simple features.

(RQ4) Can DFST evade scanning techniques

We evaluate our attack against three state-of-art backdoor scanners, ABS (Liu et al. 2019), Neural Cleanse (NC) (Wang et al. 2019), and ULP (Kolouri et al. 2020). Our results show that none of them is effective to detect models attacked by DFST. Details can be found in the repository (Cheng et al. 2020).

(RQ5) Is DFST robust?

To study robustness, we conduct three experiments. The first is to study if the injected backdoors can survive two popular adversarial training methods FGSM (Goodfellow, Shlens, and Szegedy 2014) and PGD (Madry et al. 2017). In the second experiment, we use *randomized smoothing* (Cohen, Rosenfeld, and Kolter 2019) to study the certified (radius) bound and accuracy of a trojaned model on both the benign and the malicious samples. In the third one, we perform several spacial and chromatic transformations (Li et al. 2020b) to test the degradation of attack success rate (ASR) and check DFST's robustness against pre-processing defending. The results show that DFST is robust. Details can be found in the repository (Cheng et al. 2020).

Conclusion

We introduce a new backdoor attack to deep learning models. Different from many existing attacks, the attack is in the feature space. It leverages a process called controlled detoxification to ensure that the injected backdoor is dependent on deep features instead of shallow ones. Our experiments show that the attack is effective, relatively more stealthy than many existing attacks, robust, and resilient to existing scanning techniques.

Acknowledgments

This research was supported, in part by NSF 1901242 and 1910300, ONR N000141712045, N000141410468 and

N000141712947, and IARPA TrojAI W911NF-19-S-0012. Any opinions, findings, and conclusions in this paper are those of the authors only and do not necessarily reflect the views of our sponsors.

References

Chattopadhay, A.; Sarkar, A.; Howlader, P.; and Balasubramanian, V. N. 2018. Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 839–847. IEEE.

Chen, B.; Carvalho, W.; Baracaldo, N.; Ludwig, H.; Edwards, B.; Lee, T.; Molloy, I.; and Srivastava, B. 2018a. Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728* .

Chen, B.; Carvalho, W.; Baracaldo, N.; Ludwig, H.; Edwards, B.; Lee, T.; Molloy, I.; and Srivastava, B. 2018b. Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728* .

Chen, H.; Fu, C.; Zhao, J.; and Koushanfar, F. 2019. DeepInspect: A Black-box Trojan Detection and Mitigation Framework for Deep Neural Networks. In *IJCAI*, 4658–4664.

Chen, X.; Liu, C.; Li, B.; Lu, K.; and Song, D. 2017. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526* .

Cheng, S.; Liu, Y.; Ma, S.; and Zhang, X. 2020. DFST. URL https://github.com/Megum1/DFST.

Chou, E.; Tramèr, F.; Pellegrino, G.; and Boneh, D. 2018. Sentinet: Detecting physical attacks against deep learning systems. *arXiv preprint arXiv:1812.00292* .

Cohen, J. M.; Rosenfeld, E.; and Kolter, J. Z. 2019. Certified adversarial robustness via randomized smoothing. *arXiv preprint arXiv:1902.02918* .

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, 248–255. Ieee.

Doan, B. G.; Abbasnejad, E.; and Ranasinghe, D. C. 2019. Februus: Input purification defense against trojan attacks on deep neural network systems. In *arXiv: 1908.03369*.

Gao, Y.; Xu, C.; Wang, D.; Chen, S.; Ranasinghe, D. C.; and Nepal, S. 2019. STRIP: A Defence Against Trojan Attacks on Deep Neural Networks. *arXiv preprint arXiv:1902.06531* .

Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* .

Gu, T.; Dolan-Gavitt, B.; and Garg, S. 2017. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733* .

Guo, C.; Wu, R.; and Weinberger, K. Q. 2020. TrojanNet: Embedding Hidden Trojan Horse Models in Neural Networks. *arXiv preprint arXiv:2002.10078* .

Guo, W.; Wang, L.; Xing, X.; Du, M.; and Song, D. 2019. Tabor: A highly accurate approach to inspecting and restoring trojan backdoors in ai systems. *arXiv preprint arXiv:1908.01763* .

Gupta, R. 2020. Weather-Dataset on Kaggle (Contains 4 classes such as cloudy, rain, shine and sunrise.). https://www.kaggle.com/rahul29g/weatherdataset/notebooks?sortBy=hotness&group=everyone&pageSize=20&datasetId=737827. (Accessed on 09/08/2020).

Ilyas, A.; Santurkar, S.; Tsipras, D.; Engstrom, L.; Tran, B.; and Madry, A. 2019. Adversarial examples are not bugs, they are features. In *Advances in Neural Information Processing Systems*, 125–136.

instagram filters. 2019. instagram-filters. URL https://github.com/acoomans/instagram-filters.

Kolouri, S.; Saha, A.; Pirsiavash, H.; and Hoffmann, H. 2020. Universal Litmus Patterns: Revealing Backdoor Attacks in CNNs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 301–310.

Krizhevsky, A.; and Hinton, G. 2009. Learning multiple layers of features from tiny images. Technical report, Citeseer.

Li, Y.; Wu, B.; Jiang, Y.; Li, Z.; and Xia, S.-T. 2020a. Backdoor learning: A survey. *arXiv preprint arXiv:2007.08745* .

Li, Y.; Zhai, T.; Wu, B.; Jiang, Y.; Li, Z.; and Xia, S. 2020b. Rethinking the Trigger of Backdoor Attack. *arXiv preprint arXiv:2004.04692* .

Liao, C.; Zhong, H.; Squicciarini, A.; Zhu, S.; and Miller, D. 2018. Backdoor embedding in convolutional neural network models via invisible perturbation. *arXiv preprint arXiv:1808.10307* .

Liu, K.; Dolan-Gavitt, B.; and Garg, S. 2018. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, 273–294. Springer.

Liu, Y.; Lee, W.-C.; Tao, G.; Ma, S.; Aafer, Y.; and Zhang, X. 2019. FABS: Scanning Neural Networks for Back-doors by Artificial Brain Stimulation. In *2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19)*.

Liu, Y.; Ma, S.; Aafer, Y.; Lee, W.-C.; Zhai, J.; Wang, W.; and Zhang, X. 2017. Trojaning attack on neural networks .

Liu, Y.; Ma, X.; Bailey, J.; and Lu, F. 2020a. Reflection backdoor: A natural backdoor attack on deep neural networks. *arXiv preprint arXiv:2007.02343* .

Liu, Y.; Mondal, A.; Chakraborty, A.; Zuzak, M.; Jacobsen, N.; Xing, D.; and Srivastava, A. 2020b. A Survey on Neural Trojans. *IACR Cryptol. ePrint Arch.* 2020: 201.

Liu, Y.; Xie, Y.; and Srivastava, A. 2017. Neural trojans. In *2017 IEEE International Conference on Computer Design (ICCD)*, 45–48. IEEE.

Madry, A.; Makelov, A.; Schmidt, L.; Tsipras, D.; and Vladu, A. 2017. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083* .

Moosavi-Dezfooli, S.-M.; Fawzi, A.; Fawzi, O.; and Frossard, P. 2017. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1765–1773.

Parkhi, O. M.; Vedaldi, A.; Zisserman, A.; et al. 2015. Deep face recognition. In *bmvc*, volume 1, 6.

Qiao, X.; Yang, Y.; and Li, H. 2019. Defending neural backdoors via generative distribution modeling. In *Advances in Neural Information Processing Systems*, 14004–14013.

Rakin, A. S.; He, Z.; and Fan, D. 2020. TBT: Targeted Neural Network Attack with Bit Trojan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 13198–13207.

Rezaei, S.; and Liu, X. 2019. A target-agnostic attack on deep models: Exploiting security vulnerabilities of transfer learning. *arXiv preprint arXiv:1904.04334* .

Saha, A.; Subramanya, A.; and Pirsiavash, H. 2019. Hidden trigger backdoor attacks. *arXiv preprint arXiv:1910.00033* .

Salem, A.; Wen, R.; Backes, M.; Ma, S.; and Zhang, Y. 2020. Dynamic Backdoor Attacks Against Machine Learning Models. *arXiv preprint arXiv:2003.03675* .

Shafahi, A.; Huang, W. R.; Najibi, M.; Suciu, O.; Studer, C.; Dumitras, T.; and Goldstein, T. 2018. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *Advances in Neural Information Processing Systems*, 6103–6113.

Stallkamp, J.; Schlipsing, M.; Salmen, J.; and Igel, C. 2012. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks* (0): –. ISSN 0893-6080. doi:10.1016/j.neunet.2012.02.016. URL http://www.sciencedirect.com/science/article/pii/S0893608012000457.

Steinhardt, J.; Koh, P. W. W.; and Liang, P. S. 2017. Certified defenses for data poisoning attacks. In *Advances in neural information processing systems*, 3517–3529.

Tang, R.; Du, M.; Liu, N.; Yang, F.; and Hu, X. 2020. An embarrassingly simple approach for trojan attack in deep neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 218–228.

Tran, B.; Li, J.; and Madry, A. 2018. Spectral signatures in backdoor attacks. In *Advances in Neural Information Processing Systems*, 8000–8010.

Tsipras, D.; Santurkar, S.; Engstrom, L.; Turner, A.; and Madry, A. 2018. Robustness may be at odds with accuracy. *arXiv preprint arXiv:1805.12152* .

Turner, A.; Tsipras, D.; and Madry, A. 2018. Clean-label backdoor attacks .

Wang, B.; Cao, X.; Gong, N. Z.; et al. 2020. On Certifying Robustness against Backdoor Attacks via Randomized Smoothing. *arXiv preprint arXiv:2002.11750* .

Wang, B.; Yao, Y.; Shan, S.; Li, H.; Viswanath, B.; Zheng, H.; and Zhao, B. Y. 2019. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. *Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks* 0.

Wang, Z.; Bovik, A. C.; Sheikh, H. R.; and Simoncelli, E. P. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing* 13(4): 600–612.

Xu, X.; Wang, Q.; Li, H.; Borisov, N.; Gunter, C. A.; and Li, B. 2019. Detecting AI Trojans Using Meta Neural Analysis. *arXiv preprint arXiv:1910.03137* .

Yao, Y.; Li, H.; Zheng, H.; and Zhao, B. Y. 2019. Latent backdoor attacks on deep neural networks. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2041–2055.

Zhu, C.; Huang, W. R.; Shafahi, A.; Li, H.; Taylor, G.; Studer, C.; and Goldstein, T. 2019. Transferable clean-label poisoning attacks on deep neural nets. *arXiv preprint arXiv:1905.05897* .

Zhu, J.-Y.; Park, T.; Isola, P.; and Efros, A. A. 2017. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, 2223–2232.

Zou, M.; Shi, Y.; Wang, C.; Li, F.; Song, W.; and Wang, Y. 2018. Potrojan: powerful neural-level trojan designs in deep learning models. *arXiv preprint arXiv:1802.03043* .

Appendix

A. Details of CycleGAN

Given the two (orthogonal) input image domains, A and B, CycleGAN learns a mapping $G : A \rightarrow B$ such that $G(A) \approx B$ using adversarial loss. Besides, it also learns the inverse mapping $F : B \rightarrow A$. In order to generate high-quality mappings, it leverages two training cycles, the first enforces $F(G(A)) \approx A$ and the other $G(F(B)) \approx B$, driven by cycle consistency losses. Figure 8 shows the structure of CycleGAN, with the two cycles next to each other vertically, and proceeding in opposite directions. There are two generators, G_{A2B} and G_{B2A}, and two discriminators D_A and D_B in the structure. Suppose we have two image domains, A and B. G_{A2B} translates an image from domain A, denoted as i_A, to an image in domain B, denoted as a *translated sample* $\tilde{i}_B$, while G_{B2A} the opposite. We denote the input generated by applying G_{B2A} on $\tilde{i}_B$ as $\hat{i}_A$, called the *cyclic sample*. The two discriminators determine if a sample is in the respective domains.

Three kinds of loss functions are used in training. The first one is the typical adversary loss or GAN loss that ensures the generated samples fall into a specific domain. The second is the *cycle consistency loss* that ensure the two generators are appropriately inverse to each other. The last one is the *identity loss* which ensures that if an input in the target domain is provided to a generator, the generator has no effect on the input.

The following presents the GAN losses for the two respective generators. We use a and b to denote samples from A and B, respectively.

$$\begin{aligned}\mathcal{L}_{GAN}(G_{A2B}, D_B, A, B) &= \mathbb{E}_{b\sim\mathcal{P}_{data}(b)}[\log D_B(b)] \\ &+ \mathbb{E}_{a\sim\mathcal{P}_{data}(a)}[\log(1 - D_B(G_{A2B}(a)))],\end{aligned} \tag{1}$$

$$\begin{aligned}\mathcal{L}_{GAN}(G_{B2A}, D_A, B, A) &= \mathbb{E}_{a\sim\mathcal{P}_{data}(a)}[\log D_A(a)] \\ &+ \mathbb{E}_{b\sim\mathcal{P}_{data}(b)}[\log(1 - D_A(G_{B2A}(b)))],\end{aligned} \tag{2}$$

The cycle consistency loss is to reduce the difference between inputs and their projected versions after a cycle (i.e., after a mapping and then the inverse mapping).

$$\begin{aligned}\mathcal{L}_{cyc}(G_{A2B}, G_{B2A}) &= \mathbb{E}_{a\sim\mathcal{P}_{data}(a)}[\|G_{B2A}(G_{A2B}(a)) - a\|_1] \\ &+ \mathbb{E}_{b\sim\mathcal{P}_{data}(b)}[\|G_{A2B}(G_{B2A}(b)) - b\|_1].\end{aligned} \tag{3}$$

The identity loss is defined as follows.

$$\begin{aligned}\mathcal{L}_{id}(G_{A2B}, G_{B2A}) &= \mathbb{E}_{a\sim\mathcal{P}_{data}(a)}[\|G_{B2A}(a) - a\|_1] \\ &+ \mathbb{E}_{b\sim\mathcal{P}_{data}(b)}[\|G_{A2B}(b) - b\|_1].\end{aligned} \tag{4}$$

The overall objective function is hence defined as an aggregation of the three aforementioned losses.

$$\begin{aligned}\mathcal{L}(G_{A2B}, G_{B2A}, D_A, D_B) &= \mathcal{L}_{GAN}(G_{A2B}, D_B, A, B) \\ &+ \mathcal{L}_{GAN}(G_{B2A}, D_A, B, A) \\ &+ \alpha\mathcal{L}_{cyc}(G_{A2B}, G_{B2A}) \\ &+ \beta\mathcal{L}_{id}(G_{A2B}, G_{B2A}),\end{aligned} \tag{5}$$

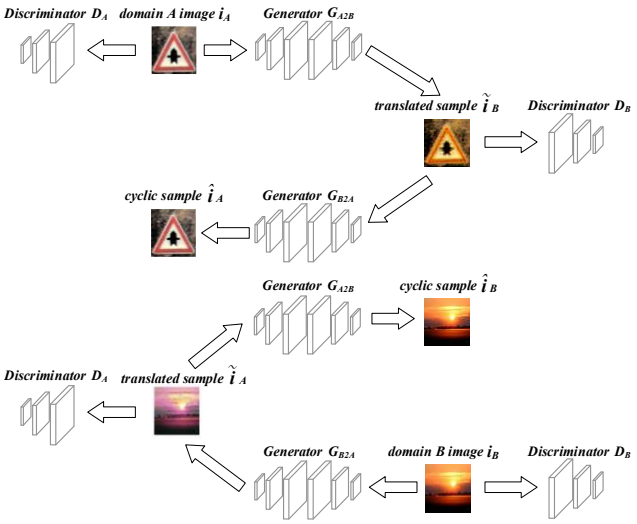

Figure 8: CycleGAN structure

Here α and β control the relative importance of three objectives, usually $\alpha = 10$ and $\beta = 1$, and we aim to solve:

$$G^*_{A2B}, G^*_{B2A} = \arg \min_{G_{A2B}, G_{B2A}} \max_{D_A, D_B} \mathcal{L}(G_{A2B}, G_{B2A}, D_A, D_B). \tag{6}$$

Intuitively, it aims to search for the generator parameters that can minimize the maximum adversarial loss.

B. Dataset Preprocessing

- CIFAR-10 (Krizhevsky and Hinton 2009) is a well-known standard image classification dataset with the image size of 32×32. It has 10 classes, with 5000 images per class for training and 1000 images per class for testing. For the initial data-poisoning, we randomly select 100 images from all 10 classes, 2% of the original training set, while for the detoxification rounds, we apply inject features to to 50 images per class (in each round), 1% of the original training set.
- GTSRB (Stallkamp et al. 2012) is another widely used image classification dataset of German traffic signs. It contains 43 classes and about $3.7k$ images. While they are not of the same size, we resize the images to 48×48. Similarly, we take 100 images per class for data-poisoning and 50 images per class for detoxification.
- VGG-Face (Parkhi et al. 2015) is a common face recognition dataset that contains $2,622$ identities with 1000 photos each. We conduct our experiments based on a subset of 20 labels with 500 images per label. Similarly, we resize the images to 224×224 and take 50 images per class for data-poisoning and 10 images per class for detoxification.
- ImageNet (Deng et al. 2009) is a large object recognition dataset, containing over 15 millions high-resolution (224×224) images in roughly 22,000 categories. We use

Table 3: Neural Cleanse result

Dataset	Model	Anomaly Index	Detected
CIFAR-10	NiN	1.629	×
	VGG	1.853	×
	ResNet32	1.790	×
GTSRB	NiN	1.853	×
	VGG	1.529	×
	ResNet32	1.918	×
VGG-Face	VGG	1.440	×
	ResNet50	1.798	×
ImageNet	ResNet101	1.308	×

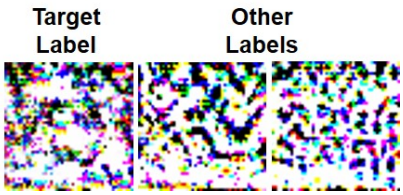

Figure 9: Trigger pattern examples by Neural Cleanse. The first is the target label trigger pattern and the other two are for other labels. Note that there is not much size difference.

a subset with 10 classes and 1000 images per class. We use 50 images per class for data-poisoning and 20 images per class for detoxification.

C. (RQ4) Can DFST evade scanning techniques

We evaluate our attack against three state-of-art backdoor scanners, ABS (Liu et al. 2019), Neural Cleanse (NC) (Wang et al. 2019), and ULP (Kolouri et al. 2020).

Neural Cleanse (NC). As mentioned in the introduction, NC uses optimization to find a universal perturbation pattern (called *trigger pattern*) for each output label such that applying the pattern to any sample causes the model to classify the label. It considers a model trojaned if the pattern of some label is much smaller than those of the others. It uses a metric called *anomaly index* to measure the anomaly level of a trigger pattern. If there is a label with an anomaly index larger than 2, it considers the model trojaned. Table 3 shows the NC results. Observe that none of the DFST-attacked models can be detected. The underlying reason is that the input transformation by our attack is global (i.e., on a whole image) so that the corresponding pixel space trigger pattern is very large. Figure 9 shows some examples of the generated trigger patterns.

ABS. ABS improves over NC by using a stimulation analysis to first identify neurons that demonstrate abnormal behaviors when their activation values are enlarged, and then using an optimization technique similar to NC to generate trigger, with the guidance of those neurons. It considers a model trojaned if a small trigger can be generated. It also handles simple filter attacks (e.g., triggering a backdoor by applying a Gaussian Instagram filter to a benign input) by optimizing a kernel as the trigger. In this case, it considers a model trojaned if applying the kernel (through a sliding window) to the input can cause misclassification.

Table 4: ABS results before and after detoxification. The numbers in parentheses denote the neurons that can be used to reverse engineer triggers.

Dataset	Model	Before Detoxification	After Detoxification
CIFAR-10	NiN	✓ (1)	×
	VGG	✓ (4)	×
	ResNet32	✓ (2)	×
GTSRB	NiN	✓ (3)	×
	VGG	✓ (5)	×
	ResNet32	×	×
VGG-Face	VGG	×	×
	ResNet50	×	×
ImageNet	ResNet101	×	×

Figure 10: Examples to illustrate ABS. From left to right, the original image, the image with our trigger applied, and the image with the trigger by ABS applied. Note that here ABS is applied before detoxification. The trigger by ABS hence induces misclassification.

Table 4 shows the results. Observe that before detoxification, ABS can detect a few of the trojaned models. But it can detect none after. This is because detoxification suppresses neuron abnormal behaviors. A sample trigger generated by ABS can be found in Figure 10.

ULP. ULP trains a number of input patterns from a large number of benign and trojaned models. These patterns are supposed to induce different output logits for benign and trojaned models such that they can be used for scanning. ULP is claimed to be model structure agnostic and trigger-type agnostic (Kolouri et al. 2020). Since it requires a large number of models for training, we use the TrojAI Round 1 training dataset[1] that consists of 500 benign models and 500 trojaned models (with various model structures and trigger types) to train the patterns. The training accuracy reaches 95%.

We then apply the trained ULP to the DFST-attacked VGG and ResNet50 on VGG-Face and ResNet101 on ImageNet. We cannot test on CIFAR-10 or GTSRB as the ULP is trained on high-resolution images ($224 * 224$)). Table 5 shows the results. Observe none of the DFST-attacked models can be detected. It discloses that the unique DFST attack mechanism makes the ULP *trained on existing models* ineffective. Due to the high detoxification cost, acquiring a large set of DFST attacked models for ULP training is presently infeasible, we will leave it to our future work.

D. (RQ5) Is DFST robust?

In this section, we study the robustness of DSFT. We conduct three experiments. The first is to study if the injected

[1] https://pages.nist.gov/trojai/docs/data.html

Table 5: ULP result

Dataset	Model	Detected
VGG-Face	VGG	×
	ResNet50	×
ImageNet	ResNet101	×

Figure 11: Examples of three types of backdoor attack triggers

backdoor can survive adversarial training. We apply two popular adversarial training methods *Fast Gradient Sign Method* (FGSM) (Goodfellow, Shlens, and Szegedy 2014) and *Projected Gradient Descent* (PGD) (Madry et al. 2017) to the 6 trojaned and detoxified models on CIFAR10 and GTSRB. In the adversarial training, we preclude all the malicious or detoxicant samples and only start with the original benign samples. This is to simulate the situation in which normal users harden pre-trained models. We use the default settings for the two methods (i.e., $\epsilon = 0.2$ for FGSM and $l_\infty = 5/255$ for PGD). In the second experiment, we use *randomized smoothing* (Cohen, Rosenfeld, and Kolter 2019) to study the certified (radius) bound and accuracy of a trojaned model on both the benign and the malicious samples. In the third one, we perform several spacial and chromatic transformations (Li et al. 2020b) to test the degradation of attack success rate(ASR) and check DFST's robustness against pre-processing defense.

For comparison, we perform the two adversarial training methods on three different kinds of backdoor attacks: the pixel-space patch (or watermark) attack as in the original data poisoning paper (Gu, Dolan-Gavitt, and Garg 2017; Chen et al. 2017), the linear filter attack mentioned in ABS (Liu et al. 2019) and our DFST attack. Figure 11 shows trigger examples of the three types of attacks.

Table 6 shows the results of adversarial training. We observe that the both the watermark and DFST attacks are resilient to adversarial training, meaning the attack success rate after adversarial training does not degrade, whereas the linear filter attack has substantial (up to 58%) degradation. This seems to indicate in the linear filter attack, the trojaned models tend to learn unrobust trigger features (Tsipras et al. 2018; Ilyas et al. 2019).

Table 7 shows the randomized smoothing results on the three attacks. We only conduct the experiment on VGG-16 with GTSRB, using 3000 samples and $\alpha = 0.001$. We study multiple Gaussian noise variance σ settings. For each setting, we report the certified radius R and the accuracy for smoothed classifier g on both the benign and the malicious samples. From the results, when σ is normal, the certified radius R for malicious samples tends to be smaller than that of benign samples but the differences are within a normal range. The accuracy is almost equally high. This demonstrates the robustness of these attacks, including DFST, in the context of random smoothing.

Table 8 shows the results of pre-processing defense (Li et al. 2020b) on DFST attacked models. We conduct experiments on VGG and ResNet32 with GTSRB using 6 transformation algorithms, Flip (flipping), ShrinkPad (random padding after shrinking with the parameters meaning the number of shrunk pixels), Gaussian (adding Gaussian noise with 0 mean and some std values), Brightness (changing brightness), Saturation (changing saturation), and Contrast (changing contrast). Note that we change the images' brightness, saturation and contrast using the image enhancing functions in PIL (*PIL.ImageEnhance.Brightness(img).enhance(extent)*). We report the clean accuracy and attack success rate (ASR) changes on all the transformation algorithms. The results show that DFST is robust as ASR does not degrade in any transformation even when the clean accuracy degrades.

Table 6: Attack robustness after adversarial training. "Benign Test" presents the accuracy of benign inputs and "Attack Test" presents the attack success rate on samples with triggers.

Dataset	Model	Trigger	Without Adv Training		PGD L-inf 5/255		FGSM Stride 0.2	
			Benign Test	Attack Test	Benign Test	Attack Test	Benign Test	Attack Test
CIFAR10	NiN	Watermark	0.916	0.988	0.869	0.996	0.913	0.992
		Linear Filter	0.913	0.963	0.864	0.867	0.911	0.894
		DFST	0.899	0.970	0.855	0.963	0.902	0.973
	VGG	Watermark	0.925	0.999	0.878	0.999	0.918	1.000
		Linear Filter	0.925	0.928	0.879	0.633	0.918	0.848
		DFST	0.928	0.999	0.858	0.990	0.902	0.985
	ResNet	Watermark	0.921	0.994	0.839	0.999	0.913	0.992
		Linear Filter	0.921	0.960	0.836	0.843	0.911	0.910
		DFST	0.900	0.976	0.827	0.943	0.889	0.972
GTSRB	NiN	Watermark	0.984	1.000	0.964	1.000	0.980	1.000
		Linear Filter	0.969	0.994	0.966	0.424	0.975	0.954
		DFST	0.963	0.988	0.964	0.981	0.975	0.981
	VGG	Watermark	0.983	1.000	0.944	1.000	0.968	1.000
		Linear Filter	0.970	0.993	0.946	0.439	0.975	0.991
		DFST	0.962	0.987	0.957	0.971	0.976	0.964
	ResNet	Watermark	0.969	1.000	0.966	1.000	0.982	1.000
		Linear Filter	0.971	0.992	0.963	0.416	0.976	0.991
		DFST	0.967	0.980	0.964	0.985	0.975	0.998

Table 7: Randomized smoothing results on trojaned VGG16 on GTSRB

Dataset		$\sigma = 0.5$		$\sigma = 0.2$		$\sigma = 0.1$		$\sigma = 0.05$	
		Benign	Trojan	Benign	Trojan	Benign	Trojan	Benign	Trojan
DFST	Radius	0.198	0.304	0.292	0.160	0.237	0.297	0.270	0.163
	Accuracy	0.728	0.931	0.977	0.977	1.000	1.000	1.000	1.000
Watermark	Radius	0.175	0.247	0.238	0.151	0.763	0.291	0.195	0.143
	Accuracy	0.488	0.898	0.929	0.934	1.000	1.000	1.000	1.000
Linear Filter	Radius	0.153	0.197	0.201	0.155	0.749	0.303	0.163	0.098
	Accuracy	0.389	0.913	0.929	0.930	0.966	0.806	0.789	0.801

Table 8: Results of (Li et al. 2020b) on trojaned VGG and ResNet32 on GTSRB

Transformation	VGG		ResNet32	
	Clean	ASR	Clean	ASR
Standard	0.92	0.99	0.92	0.99
Flip	0.92	0.98	0.92	0.99
ShrinkPad-1	0.91	0.98	0.90	0.98
ShrinkPad-2	0.90	0.97	0.90	0.96
ShrinkPad-4	0.88	0.93	0.87	0.88
Gaussian-std=5/255	0.89	0.98	0.88	0.86
Gaussian-std=5/255	0.57	0.95	0.65	0.71
Brightness=0.8	0.92	0.98	0.91	0.98
Brightness=1.2	0.92	0.97	0.90	0.98
Saturation=0.8	0.92	0.98	0.92	0.97
Saturation=1.2	0.92	0.98	0.91	0.99
Contrast=0.8	0.92	0.99	0.92	0.99
Contrast=1.2	0.92	0.97	0.91	0.97