Online Defense of Trojaned Models using Misattributions

Panagiota Kiourti[1], Wenchao Li[1], Anirban Roy[2], Karan Sikka[2], and Susmit Jha[2]

[1] Boston University
[2] SRI International
{pkiourti, wenchao}@bu.edu, {anirban.roy, karan.sikka, susmit.jha}@sri.com

Abstract. This paper proposes a new approach to detecting neural Trojans on Deep Neural Networks during inference. This approach is based on monitoring the inference of a machine learning model, computing the attribution of the model's decision on different features of the input, and then statistically analyzing these attributions to detect whether an input sample contains the Trojan trigger. The anomalous attributions, aka misattributions, are then accompanied by reverse-engineering of the trigger to evaluate whether the input sample is truly poisoned with a Trojan trigger. We evaluate our approach on several benchmarks, including models trained on MNIST, Fashion MNIST, and German Traffic Sign Recognition Benchmark, and demonstrate the state of the art detection accuracy.

1 Introduction

Deep Learning has made significant progress over the last decade allowing us to tackle numerous challenging tasks [37,20]. However, these models have been shown to be fragile and susceptible to adversarial attacks during various stages of their deployment [12,40,18,47,48]. In particular, recently introduced *backdoor attacks* [12,18] allow an attacker to fully control the model's behavior during inference as shown in Fig. 1 using a *Trojan trigger* injected in the data during training.

Existing state-of-the-art defenses [44,3,31,5] against Trojan attacks analyze the network to identify whether it can be controlled by a Trojan trigger. However, these methods have high computational cost and limitations based on the number of labels [44], trigger's size or location in the image [5] and often cannot detect complex triggers such as dynamic triggers [33]. Other methods [10,6] perform run-time analysis to identify potentially Trojan triggers present in the inference-time images.

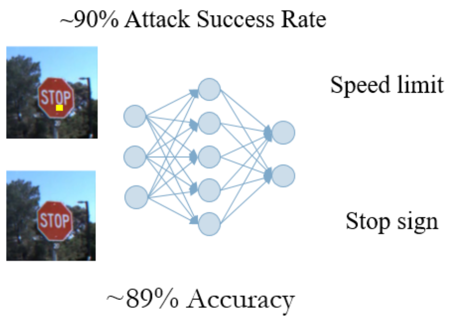

Fig. 1. An example Trojaned model [12] that misclassifies the Stop sign as Speed Limit when the Trojan is present. The model is 89% accurate with 90% Attack Success Rate.

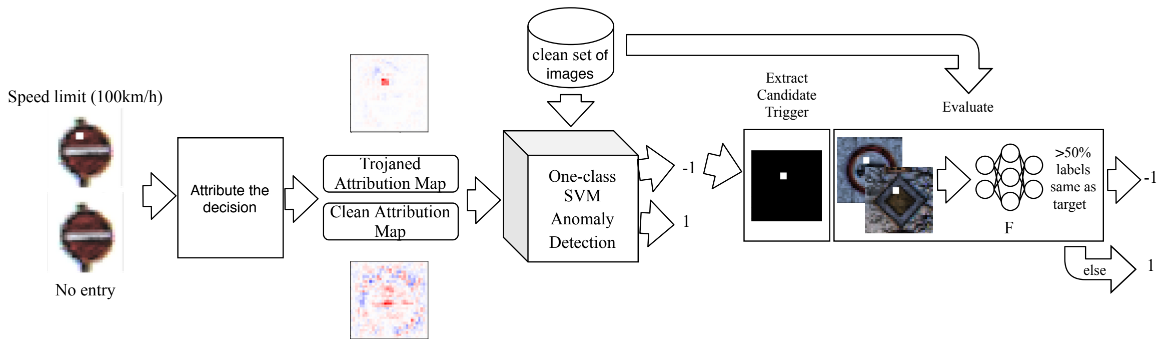

Fig. 2. Overview of our approach with 2 instances: a clean image of 'no entry' and the image Trojaned with a white square trigger.

At the same time, several works with the aim to explain the decisions of neural networks have been introduced [38,49,1,36,39,27,30,16]. These methods, often called attribution methods, try to explain the neural network's output by assigning an importance value to each input feature based on the decision taken for this input. We use the term features to refer to input values or an intermediate layer's output values. These methods rely on the gradient of the neural network's output on a given input to compute the importance values (a.k.a attributions).

This paper develops a novel approach to detect input images poisoned with triggers (hereafter referred to as Trojaned images) that correspond to a Trojaned deep neural network. Our method is a run-time online approach that monitors the input and the attributions of the model's decision on the input features. The key intuition is that a poisoned input with a trigger generates a completely different attribution for the Trojaned model's decision than its attributions over the clean inputs. Our contributions are as follows:

– We propose a novel method for monitoring the inference of a neural network at runtime and determining whether an input contains a Trojan trigger. The method leverages *attributions*, which computes the relative importance of different features when a neural network makes a prediction on a given input. We demonstrate our approach using input space features as well as the internal layers of the neural network.
– We present *misattributions*, a formal characterization of anomaly that manifests in the unusual attributions of features when a Trojan trigger is present in an input. In particular, misattributions hold high importance values for input features not expected to be high for a given label.
– With extensive experiments on different types of triggers and datasets, we demonstrate that our method can effectively detect the presence of a Trojan trigger without assuming any prior knowledge of the trigger.

2 Related Work

Adversarial Attacks. There has been a significant investigation of adversarial examples and defenses [13,21] that highlight the vulnerability of deep learning models to adversarial perturbations. Universal adversarial attacks [28,29,42] have also been studied. Contrary to these attacks, in Trojan or Backdoor attacks [12], the DNN is trained to associate a specific pattern (Trojan trigger) in the input with a target label. The attack is insidious since the Trojan trigger is only known to the attacker; the model outputs the correct label when the trigger is absent. Other state-of-the-art Trojan insertion methods are proposed in [9,22,34,51,4]. Inserting Trojans using transfer learning [46] or retraining [25] has been demonstrated. Further, recent work show how Trojans can be inserted via the direct manipulation of DNN weights [8,32], or the addition of malicious modules [41].

Offline Trojan Defenses. Offline methods analyze the DNN model to determine whether it is Trojaned. One approach uses pruning and fine-tuning of the DNN [24]. Another approach repairs DNN based on the mode connectivity technique [52,11]. A class of defenses, pioneered in Neural Cleanse [44] and followed by [3,15,53] focuses on reverse-engineering the Trojan triggers. These methods has been shown to produce patterns distinct from the ones used in training [31]. Reverse-engineering of triggers using GANs has also been proposed [3,31,53]. Finally, recent work has focused on model diagnosis approaches. These include the use of universal litmus test [19], differential privacy [7], one-pixel signature [14], and a combination of adversarial attacks and feature inversion [45,50]. On the contrary, we develop an online inference-time defense.

Online Trojan Defenses. Online methods filter inputs of a DNN to detect poisoned samples during inference. The inference-time pre-processing of the samples for detecting poisoned inputs to a potentially Trojaned DNN model has also been investigated in the literature. One approach to pre-processing uses pre-trained autoencoders [26]. Style transfer [43] and spatial transformations such as shrinking and flipping [23] have also been shown to be effective against existing Trojan attacks. Another approach uses a superimposition of various image patterns followed by observing the conformance of prediction [10]. As demonstrated by the US NIST-TrojAI datasets, injected triggers can be made robust to many kinds of transformations. Thus, these approaches rely on the fragility of the trigger and are not sufficient. In contrast, our approach relies on using attributions to analyze deviation in what features are important for inputs. Thus, it presents a more principled and robust approach to detecting Trojaned samples during inference.

3 Preliminaries

Trojan Trigger. Let $\boldsymbol{x}$ be a d-dimensional input and y be the class label coming from a data distribution D. We consider a neural network F for image classification such that $F(\boldsymbol{x})$ is the predicted class for $\boldsymbol{x}$. For a Trojaned model, we define a Trojan trigger as $\boldsymbol{\delta} \in \mathcal{R}^d$ such that

$$P_{(\boldsymbol{x},y)\sim D}[F(\boldsymbol{x}) = y] \geq 1 - \epsilon_1, \text{ and} \tag{1}$$

$$P_{(\boldsymbol{x},y)\sim D}[F(\widetilde{\boldsymbol{x}}) \neq F(\boldsymbol{x})] \geq 1 - \epsilon_2, \tag{2}$$

where $\widetilde{\boldsymbol{x}} = \boldsymbol{x} \oplus \boldsymbol{\delta}$ is the Trojaned input that corresponds to the *injection* of the Trojan trigger $\boldsymbol{\delta}$ into the input $\boldsymbol{x}$. ϵ_1 and ϵ_2 are small numbers. Intuitively, the definition implies that a Trojaned model is expected to flip the decision for Trojaned inputs $\widetilde{\boldsymbol{x}}$ with high probability but correctly predicts the labels for non-Trojaned inputs. We consider two types of Trojans: 1) *Patch-based triggers* where $\widetilde{\boldsymbol{x}}$ is computed as $\widetilde{\boldsymbol{x}}_i = (1 - m_i) \cdot \boldsymbol{x}_i + m_i \cdot \delta_j$, $\widetilde{\boldsymbol{x}}_i$ refers to the

ith element and $m \in \mathcal{R}^d$ is the associated, sparse mask of the trigger. 2) *Image-based triggers* where the trigger is applied to an entire image, i.e., $\widetilde{\boldsymbol{x}}_i = \boldsymbol{x}_i + \delta_i$.

Attributions. Following the definition in [39], given a neural network F and an input $\boldsymbol{x} \in \mathcal{R}^d$, an attribution of a prediction against a baseline $\boldsymbol{x}^b$ is a vector $\boldsymbol{a} \in \mathcal{R}^d$, where $\boldsymbol{a}_i$ represents the contribution of $\boldsymbol{x}_i$ towards the prediction. In this paper, we compute attribution based on Integrated Gradients (IG) [39] as: $IG_i(\boldsymbol{x}, y) = (\boldsymbol{x}_i - \boldsymbol{x}_i^b) \times \int_{\alpha=0}^{1} \frac{\partial F^y(\boldsymbol{x}^b + \alpha \times (\boldsymbol{x} - \boldsymbol{x}^b))}{\partial \boldsymbol{x}_i} d\alpha$, where the gradient of model output corresponding to class y along the i-th input is denoted by $\partial F^y / \partial \boldsymbol{x}_i$. Intuitively, attributions provide an estimate of relative importance of an individual input component $\boldsymbol{x}_i$. We also refer to attributions on $\boldsymbol{x}$ as an attribution map in the rest of the paper.

4 Formalizing Misattributions

In this section, we formalize the concept of *misattributions*. Given a network F, we can extract the attribution map $att(\boldsymbol{x})$ for an input $\boldsymbol{x}$ and a predicted label y as $att(\boldsymbol{x}) = IG(\boldsymbol{x}, y)$. Let's denote f as the features for an input where the features can represent the raw pixels in the input space or the output of an intermediate layer, and the corresponding attribution map over f as att_f.

Misattribution: We define an *in-distribution* P_{att_f} for attributions over f on clean, labeled data. For an input $\widetilde{\boldsymbol{x}}$, its attribution over f is deemed a *misattribution* if $att_f(\widetilde{\boldsymbol{x}})$ is not from P_{att_f}. In other words, $\widetilde{\boldsymbol{x}}$ is considered as *out-of-distribution* (OOD) from the clean data in terms of its attributions over f. In the case of a Trojan trigger, $att_f(\widetilde{\boldsymbol{x}})$ is likely to be related to Trojaned predictions. In addition, for a Trojan trigger to be effective across different images, the high-attribution features $f_h \subset f$ (which correspond to class-specific discriminative features [35,2]) should satisfy the following property.

Persistently OOD: For a Trojaned image $\widetilde{\boldsymbol{x}}$ with predicted label y, the network should have a high probability of predicting y even if we replace the low-attribution features with values from P_{att_f}, that is,

$$P(F(att_f[\![f_l \setminus f_h]\!]) = y) \geq 1 - \epsilon,$$

where $f_l = f \setminus f_h$ are the low-attribution features , $att_f[\![f_l \setminus f_h]\!]$ represents an attribution map over f that is consistent with the attributions on f_h but can vary on f_l according to P_{att_f}, and ϵ is a small threshold. For instance, if f represents the raw pixels in the input, for an OOD $\widetilde{\boldsymbol{x}}$ with label y, replacing the low-attribution pixels with pixels in the same locations from another image in the data set would still likely cause F to predict y.

5 Attribution-Based Trojan Detection

Using the notion of misattributions described above, we develop a method to detect whether the input provided to a neural network during inference is Trojaned or not. Our method doesn't assume any prior knowledge of the attack, that is, we don't know the type of the Trojan trigger in advance or the target label. Also, the defender doesn't own sample Trojaned images or any reference Trojaned networks. Our method only requires access to

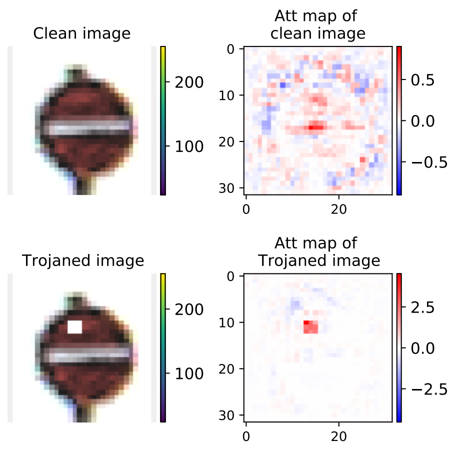

Fig. 3. Comparison of attribution maps in terms of important features for an image with a small square trigger and the corresponding clean image. The trigger region is associated with very high attributions in the Trojaned image.

the neural netwokr and a set of validation clean inputs used for evaluating the potentially Trojaned model. Our method is based on detecting outliers in the attribution space. Given a potentially Trojaned neural network $\widetilde{F}$, we observe that a Trojaned input's attribution map is a *misattribution*. Hence, it is out of the attribution maps' distribution for clean inputs (inputs without the Trojan trigger injected to them). This can be verified by observing the attributions of an input and the corresponding Trojaned attribution map, as shown in Fig. 3. Therefore, we can use an outlier detection method based on the input's attributions to identify potentially Trojaned inputs.

We use a one-class SVM to perform the anomaly detection, which is part of our online detection approach. We assume that a set of clean inputs is given to the defender for validation, which we use to compute the clean inputs' attribution maps. We train the one-class SVM on these clean inputs' attribution maps. Therefore, the SVM learns to recognize clean attribution maps as valid and Trojaned attribution maps as invalid.

Choosing the right training parameters of the one-class SVM is critical for the performance of our method. The ν parameter represents the percentage of outliers we expect to see in the set of clean inputs, used for training. This means that we expect to have ν% of clean attribution maps classified as Trojaned/invalid. Naturally, we would choose a small value for ν. However, depending on the Trojaned trigger, the attributions might look similar to the attributions of a clean image, especially when the Trojaned trigger is present in spaces where we usually have high attribution values for clean images. In the case of higher ν, the SVM will detect Trojaned inputs but will have a high false-positive rate.

Parameter γ represents the margin, which is the minimal distance between a point in the training set and the separating hyperplane that separates the training set. Higher values of γ correspond to a smaller margin. Therefore, we consider high values for γ ($\gg 0.01$) because support vectors for the clean and Trojaned attribution maps can be close to each other depending on the type of trigger.

Our end-to-end detection method includes 2 steps. First, we use the one-class SVM to identify a potential Trojaned input as shown in Fig. 2. Only in the case of identifying a potentially Trojaned image, we proceed to evaluate whether the current input flagged by the SVM as Trojaned is actually a Trojaned input or a False Positive.

First step: For a given neural network $\widetilde{F}$ and inference-time input $\boldsymbol{x}$, we compute the attribution map of the input using the current decision label $\widetilde{y}$. Suppose that the one-class SVM flags the input's attribution map as Trojaned/invalid (-1). In that case, we proceed to the 2nd step.

Algorithm 1 Extract Candidate Triggers

```
Input: inference-time image $\boldsymbol{x}$, attribution map $att(\boldsymbol{x})$ of image $\boldsymbol{x}$.
Output: list of 2 reverse-engineered triggers.
 1: $\mu \leftarrow mean(att(\boldsymbol{x}))$ // mean of $att(\boldsymbol{x})$
 2: $\sigma \leftarrow SD(att(\boldsymbol{x}))$ // standard deviation of $att(\boldsymbol{x})$
 3: $triggers \leftarrow []$
 4: for $c \in [2, 3]$ do
 5:     $mask \leftarrow \mathbf{0}$
 6:     for $i \in height(att(\boldsymbol{x}))$ do
 7:         for $j \in width(att(\boldsymbol{x}))$ do
 8:             if $att(\boldsymbol{x})[i, j] > \mu + c \cdot \sigma$ then
 9:                 $mask[i, j] \leftarrow 1$
10:             else
11:                 $mask[i, j] \leftarrow 0$
12:     $trigger \leftarrow mask \circ \boldsymbol{x}$ // element-wise product
13:     $triggers.append(trigger)$
14: return $triggers$
```

Second Step: We extract the candidate trigger using the significantly higher values of the image's attribution map (Alg. 1). We then evaluate if the current input is indeed a Trojaned input and not a false positive by injecting this candidate trigger to 100 images that don't already belong to the current label $\widetilde{y}$ (Alg. 2). We refer to these 2 steps of extracting and injecting the candidate trigger as *cut-and-paste*. We measure this trigger's ability to flip the labels of these images to the candidate target label. Suppose the candidate trigger can flip more than 50% of the labels from this set of inputs; we then consider this trigger to be a Trojaned trigger and the input to be Trojaned. Our exact algorithm is presented in Alg. 3.

6 Experiments

We evaluate our method on multiple Trojaned models that we train on MNIST, Fashion MNIST, and the German Traffic Sign Recognition Benchmark (GTSRB) (Table 1). The training hyperparameters are fixed per dataset while Trojaned models are trained to respond to different static or dynamic triggers. We keep the models that achieved Attack Success Rate (ASR) $> 90\%$ (i.e. percentage of Trojaned images classified as the target label). We refer to triggers that are always injected in the same location of the input as static triggers. When we sample the trigger and its location from a set of triggers and locations, respectively, we call the trigger dynamic [33]. More details about the triggers can be found in the Appendix.

Algorithm 2 Evaluate Trigger

Input: potential backdoored network $\widetilde{F}$, set of clean images S, candidate trigger t, candidate target label $\widetilde{y}$.
Output: Percentage of images that are labeled with the target label l when the candidate trigger t is injected to them.

```
1: $flipped \leftarrow 0$
2: $K \leftarrow$ randomly pick 100 images from $S$ (not of label $\widetilde{y}$)
3: for $im \in K$ do
4:     $y \leftarrow \widetilde{F}(im + \boldsymbol{t})$
5:     if $y == \widetilde{y}$ then
6:         $flipped \leftarrow flipped + 1$
7: return $\frac{flipped}{|K|}$
```

Algorithm 3 Detection

Input: neural network $\widetilde{F}$, SVM Model M, inference-time image $\boldsymbol{x}$, threshold th, clean set of images S
Output: -1 or 1 indicating whether the current input included a trojan or not respectively. In the case of a trojaned image, the reverse-engineered trigger is returned.

```
1: $\widetilde{y} \leftarrow \widetilde{F}(\boldsymbol{x})$ // Candidate target label
2: $att(\boldsymbol{x}) \leftarrow IG(\boldsymbol{x}, y)$ // Attribution map
3: $status \leftarrow M(att(\boldsymbol{x}))$ // SVM output
4: if $status == -1$ then
5:     $T \leftarrow$ extract_candidate_triggers($\boldsymbol{x}$, $att(\boldsymbol{x})$) // Alg. 1
6:     for $\boldsymbol{t} \in T$ do
7:         $flipped \leftarrow$ evaluate_trigger($\widetilde{F}$, $K$, $\boldsymbol{t}$, $\widetilde{y}$) // Alg. 2
8:         if $flipped \geq th$ then
9:             return $-1, \boldsymbol{t}$
10:    return 1
11: else
12:    return 1
```

Our experiments focus on studying the effect of static and dynamic triggers on our method's ability to detect Trojans and comparing our approach with current state-of-the-art methods. We also investigate how the size or the location of the trigger affects detection accuracy. In addition, we show how our method can be improved by using attributions at intermediate layers of the neural network instead of using the ones at the input layer. Finally, we present two ablation studies to further examine the contributions of the key components in our method.

Table 1. Details of our models with static/dynamic triggers.

		# models	Accuracy	ASR
Static	MNIST	102	99.1	98.8
	Fashion MNIST	78	91.3	97.7
	GTSRB	129	93.3	98.6
Dynamic	MNIST	9	99.0	99.4
	Fashion MNIST	9	90.1	95.9
	GTSRB	12	93.3	98.0

We compute attributions using DeepSHAP [27] against a black image as the baseline for MNIST and Fashion MNIST. For GTSRB, we use the evaluation set of images as baselines.

We train a one-class SVM for each Trojaned model using a Gaussian kernel with parameters ν set to 0.4, 0.6, and 0.3 and parameter γ set to $0.3, 0.2$, and 0.2 for MNIST, Fashion MNIST and GTSRB, respectively. Details about how to select the SVM parameters ν, γ can be found in the Appendix. We use the following metrics to evaluate our online detection approach:

– *SVM Acc*: the percentage of clean images identified as clean by the anomaly detection component, represented by the SVM. It determines the number of False Positives our anomaly detection step reports.
– *SVM DA*: the Detection Accuracy, i.e., the percentage of Trojaned images identified as Trojaned by the SVM.
– *Final Acc*: corresponds to our online detection's method total accuracy to identify the clean images after the cut-and-paste step. This metric shows the ability of our method to reduce False Positives determined by the SVM.
– *Final DA*: our method's total accuracy in identifying the Trojaned images after the cut-and-paste step.

In general, achieving high Final Acc and Final DA is desired so that a method can identify both clean and Trojaned images with high accuracy. Our method prioritizes on achieving a higher Final DA to ensure the detection of most if not all Trojaned instances.

6.1 Comparison with State-of-the-Art

This section presents the results of our approach averaged over all models of static or dynamic triggers. In Table 2, we show that our method detects Trojans with overall accuracy of 95.8% and 98.9% for static and dynamic triggers, respectively. The False Positive Rate that can be inferred from the Final Acc is 13.6% and 13.2%. Additionally, our method's run-time per input image is on average 92.9 ms, 125.3 ms, and 212.6 ms for MNIST, Fashion MNIST, and GTSRB, respectively. Due to the relatively more expensive computation of attributions, our method's run-time is ~ 91 times longer than the inference-time on the same input.

Table 2. Results against STRIP using dynamic and static triggers. Our method's results correspond to intermediate layer attributions.

			STRIP	Ours
Static	MNIST	Final DA	75.2 ± 31.8	$\mathbf{90.8} \pm 23.3$
		Final Acc	92.2 ± 1.0	$\mathbf{99.6} \pm 0.4$
	Fashion MNIST	Final DA	86.4 ± 25.7	$\mathbf{97.6} \pm 4.2$
		Final Acc	$\mathbf{95.2} \pm 1.6$	72.1 ± 5.2
	GTSRB	Final DA	79.1 ± 34.5	$\mathbf{99.1} \pm 3.4$
		Final Acc	$\mathbf{98.2} \pm 0.5$	87.4 ± 2.3
Dynamic	MNIST	Final DA	93.5 ± 3.3	$\mathbf{98.4} \pm 2.1$
		Final Acc	90.2 ± 1.0	$\mathbf{93.6} \pm 1.1$
	Fashion MNIST	Final DA	92.2 ± 5.6	$\mathbf{98.5} \pm 1.1$
		Final Acc	$\mathbf{93.9} \pm 0.6$	79.0 ± 3.2
	GTSRB	Final DA	49.2 ± 28.8	$\mathbf{100.0} \pm 0.1$
		Final Acc	$\mathbf{98.4} \pm 0.5$	87.8 ± 1.6

We compare our method against STRIP [10], the state-of-the-art online detection method. STRIP also relies on the persistence of the target label to detect when the input includes a Trojan trigger. Overall, our method significantly outperforms STRIP, as ours has a much higher Final DA in all cases of static and dynamic triggers. From Table 2, we can observe that STRIP misses $\sim 20\%$ (or more due to the high standard deviation) of the Trojaned images and even $\sim 50\%$ of them for dynamic triggers (GTSRB). On the other hand, STRIP shows a lower False Positive Rate (FPR) compared to our method as it has the higher Final Acc. However, a lower FPR is less important here since STRIP misses a significant portion of the Trojaned images.

STRIP detects Trojaned images by first superimposing (adding) the run-time image with each image from the evaluation set, and then observing the model's predictions on the resulting images. Their main idea is that the entropy distribution of these predicted labels for Trojan images would be significantly different from that of clean images. In our experiments, we observe that STRIP does not reliably detect Trojaned images, as evidenced by its lower Final DA with high standard deviation. Additionally, the entropy distribution of predicted labels from a Trojaned image can look similar to the one from clean images in some cases, which is also observed by [33] for dynamic triggers.

Other methods, such as [44,15,3,31] are offline methods and have a high computational cost. For example, Neural Cleanse [44] and its variant [15] require solving an expensive optimization problem per target label. For DNNs with a large number of labels, their detection method can take hours or even days [44]. Hence, we do not compare our online detection method directly with offline methods in this paper.

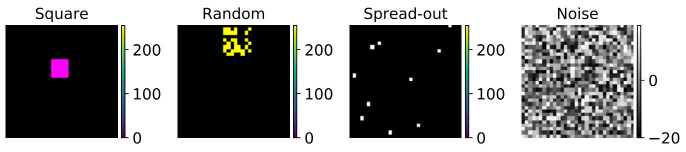

Fig. 4. A sample of different static triggers used during training.

6.2 Evaluation on Static Triggers

In this section, we study the effect of the size and the location of the trigger on the detection accuracy. We also present our method's ability to detect Trojans when intermediate-layer attributions are used instead of input-layer attributions. A sample of the used Trojaned triggers can be found in Fig. 4. We consider triggers that are not localized (i.e. not concentrated in one place of the input), image-based triggers (adding noise to the image), and localized triggers with different sizes, positions, and colors. We use the TrojAI Open Source tool [17] to create localized triggers. For spread-out, image-based triggers, we create our own triggers. More details can be found in the Appendix.

Trigger Size. Table 3 shows the results of each metric averaged over models of a specific trigger size. We observe that (i) the Final DA drops as we increase the trigger's size for all models and (ii) the Final DAs for MNIST and Fashion MNIST models are lower than those from GTSRB. Observation (ii) has to do with the attribution method's ability to derive a map close to the true explanation for the images with an artificial black background: MNIST and Fashion MNIST images. More specifically, we use a black image as the baseline image

Table 3. Detection Accuracies for all Trojaned Models in terms of the static trigger's size.

	Trigger Size		SVM		Final	
			Acc	DA	Acc	DA
MNIST	3x3	Mean	55.0	99.0	97.9	92.8
		SD	4.0	3.7	1.9	22.7
	5x5	Mean	53.6	96.5	99.1	79.9
		SD	2.5	12.2	0.9	35.7
	8x8	Mean	53.6	93.6	99.1	68.3
		SD	2.5	15.5	1.4	43.2
Fashion MNIST	3x3	Mean	39.7	99.5	100.0	75.8
		SD	1.0	1.6	0.1	34.7
	5x5	Mean	38.7	99.8	99.9	70.0
		SD	1.2	0.4	0.2	40.7
	8x8	Mean	39.3	98.5	100.0	59.0
		SD	1.3	5.3	0.1	45.9
GTSRB	3x3	Mean	99.6	99.6	97.7	99.6
		SD	1.4	1.4	1.2	1.4
	5x5	Mean	96.8	96.8	97.7	96.8
		SD	14.9	14.9	1.3	14.9
	8x8	Mean	90.2	90.2	97.6	82.1
		SD	21.0	21.0	1.2	28.8

against which we compute the attributions of MNIST and Fashion MNIST images. The implication is that any black feature of the image that contributed to the decision would be incorrectly given a 0 attribution value, as shown in Fig. 5 and also observed in [16]. Filtering

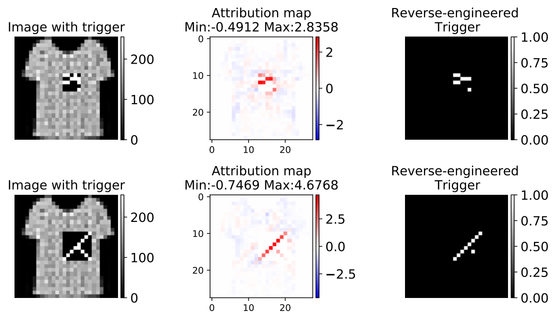

Fig. 5. Reverse-engineered triggers from attributions computed against a black baseline image. The black background of the original Trojan trigger is not reverse-engineered.

out the cases where the black background is part of the trigger (as in Fig. 5) gives more representative results. In particular, Final DAs for MNIST become 99.3% ± 1.7, 88.8% ± 26.6 and 80.0% ± 37.3 for trigger sizes 3x3, 5x5, and 8x8, respectively. Similarly, Final DAs for Fashion MNIST become 98.5% ± 3.8, 98.7% ± 2.5 and 76.4% ± 39.5 for trigger sizes 3x3, 5x5, and 8x8, respectively. Using the above filtering we look into observation (i) and see that larger triggers of a uniform color cannot be fully reverse-engineered using the input layer's attributions as shown in Fig. 6. However, this is true only for 8 (out of 33) MNIST models, 8 (out of 27) Fashion MNIST models and 9 (out of 36) GTSRB models that respond to an 8x8 trigger. In the more realistic case of GTSRB, we can detect Trojaned images with an average accuracy of 92.8% and a standard deviation of 20.2%. The high standard deviation is a result of the 9 models for the 8x8 Trojan trigger which have a lower Final DA.

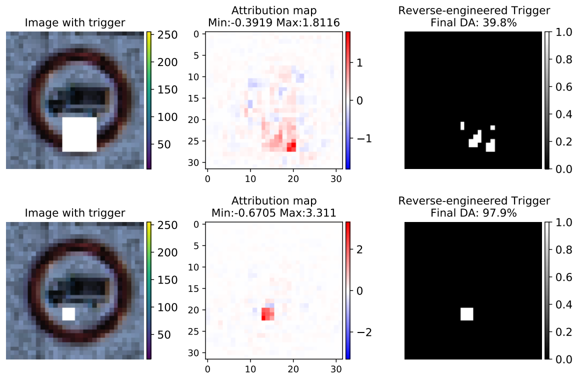

Fig. 6. Comparison of reverse-engineered triggers for large and small square triggers. Top row: image with 8x8 white square trigger, attribution map, and reverse-engineered trigger. Second row: image with 3x3 white square trigger, attribution map, and reverse-engineered trigger.

Trigger Location. In this section, we consider averaging each metric over models of a specific trigger location (Table 4). We exclude large triggers (i.e. triggers of size 8x8) in this study to better isolate the effect of varied locations. Our results suggest that triggers close to the center of the image can cause a small drop in the Final DA, since these triggers overlap with the important parts of the image. However, the reduction is not significant, as shown for GTSRB.

Next, for MNIST and Fashion MNIST, we filter out the cases of triggers with a black background. The updated Final DAs are: (MNIST) 94.3%, 77.0%, and 94.6% for TM, M, and BR respectively; (Fashion MNIST) 84.5%, 80.2%, and 94.8% for TM, M, and BR, respectively. From the updated numbers we observe that the drop in performance still exists when injecting triggers in the center. We find that this happens when the trigger is in the center of the image and its color is close to the color of the baseline image (e.g., gray (100) triggers in the center of the images are not assigned high attribution values. Hence they are not detected, as shown in Fig. 7; however, white (255) triggers are reverse-engineered).

Lastly, for GTSRB we also experiment with spread-out triggers of 9, 10, and 25 pixels as well as triggers that add noise to the whole image. Noise triggers show an average of Final Acc 93.6% $\pm$ 0.7 and Final DA of 98.6% $\pm$ 1.3. However, spread-out triggers show an average Final Acc of 99.6% $\pm$ 0.3 and Final DA of 51.8% $\pm$ 27.5. The lower Final DA in the case of spread-out triggers is a result of attributing other features than the spread-out trigger.

Intermediate Layer Attributions. The results based on the input-layer attributions are successful but also show a lower DA in some cases: spread-out triggers, large triggers,

Table 4. Detection Accuracies for all Trojaned Models in terms of the trigger's location. TM: Top Middel, M:Middle, BR: Bottom Right.

	Trigger Location		SVM		Final	
			Acc	DA	Acc	DA
MNIST	TM	Mean	53.9	99.6	98.7	97.9
		SD	3.8	1.0	1.4	5.1
	M	Mean	55.1	93.9	98.7	62.8
		SD	2.8	15.2	1.6	43.1
	BR	Mean	53.4	99.9	98.3	99.3
		SD	2.9	0.4	1.6	1.8
Fashion MNIST	TM	Mean	38.7	99.9	99.9	80.6
		SD	1.0	0.2	0.2	30.6
	M	Mean	38.9	99.8	99.9	48.4
		SD	1.2	0.5	0.1	43.1
	BR	Mean	39.4	99.5	99.9	91.0
		SD	1.4	1.6	0.1	22.8
GTSRB	TM	Mean	58.3	100.0	98.1	100.0
		SD	2.9	0.0	1.3	0.1
	M	Mean	55.4	96.7	97.8	96.7
		SD	8.6	14.9	1.1	14.9
	BR	Mean	54.5	99.4	97.3	99.4
		SD	4.3	2.2	1.4	2.2

triggers that overlap with features in the center of the image. Therefore, we can instead attribute the decision to an intermediate convolutional or max-pooling layer than the input layer to extract Trojaned-related features in the feature space. This is motivated by the fact that some Trojaned images' attributions in the input space differ from the true intuitive explanation. We expect attributions of clean and Trojaned images to differ in the feature space of an intermediate layer, where the one-class SVM will act as an anomaly detection component.

We considered attributing the decision to intermediate layers of 9216, 6272, and 2048 features for MNIST, Fashion MNIST, and GTSRB, respectively. In this case, every step of our method, summarized in Fig. 2, uses the intermediate layer's attributions. We choose convolutional or max-pooling layers in a higher-dimensional space that hold richer information and avoid layers very close to the softmax layer. However, MNIST is a small network with not a lot of hidden layers to choose from. Hence, we choose the layer of 9216 features to train the SVM.

Table 5. Detection Accuracies for all Trojaned Models based on intermediate layer attributions.

		SVM		Final	
		Acc	DA	Acc	DA
MNIST	Mean	57.7	96.9	99.6	90.3
	SD	3.0	11.7	0.4	24.2
Fashion MNIST	Mean	38.6	99.2	72.8	97.7
	SD	1.2	1.7	5.5	4.0
GTSRB	Mean	68.8	99.6	87.4	99.1
	SD	1.2	2.0	2.2	3.3

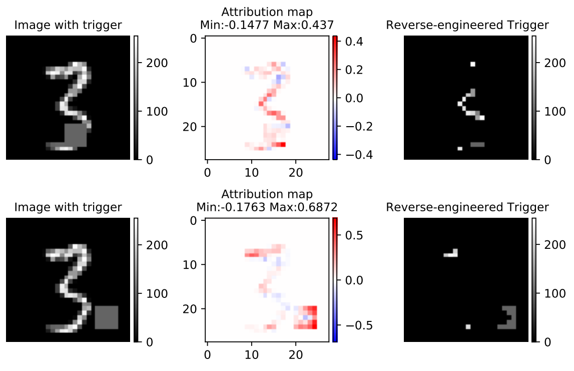

Fig. 7. Comparison of reverse-engineered triggers for different positions of the same trigger. Top row: image with trigger in the center. Second row: Image with trigger in the bottom right.

Using an intermediate layer to detect Trojans shows a significant improvement for the GTSRB models with spread-out triggers and triggers of size 8x8 for which our method show lower DA. The Final DA is 99.7% and 97.0%, respectively. Table 5 shows the accuracies averaged over all models when we use the feature space to detect Trojans. All 102 MNIST and 78 Fashion MNIST Trojaned models can be detected with 90.3% and 97.7% accuracy. As expected, there exists a few MNIST cases where the SVM's DA is lower (it can be inferred by the SD of 11.7). This also affects the Final DA of the models. We find that intermediate-layer attributions work better in identifying a variety of Trojaned models.

6.3 Evaluation on Clean Models

This section presents results on clean models, i.e. models with no injected triggers during training. It is important to evaluate whether our method flags images provided to clean models as Trojaned. Results show that our method successfully flags 99.8%, 100% and 98.3% of the reverse-engineered triggers as clean for MNIST, Fashion MNIST, and GTSRB models, respectively. There are no Trojaned triggers for computing the Final DA of our method.

Table 6. Detection on clean models trained with no triggers.

Dataset	Acc	SVM		Final	
		Acc	DA	Acc	DA
MNIST	99.1	55	NA	99.8	NA
Fashion MNIST	92.1	40	NA	100	NA
GTSRB	99.3	56.7	NA	98.3	NA

6.4 Ablation Studies

This section examines our approach's ability to detect Trojans when (1) we remove the anomaly detection component (SVM) or (2) we only utilize an SVM trained on raw images

Table 7. Ablation study that uses Grad-CAM at every step to reverse-engineer the Trigger. We use a GTSRB model that responds to a square trigger.

Grad-CAM				Ours			
SVM		Final		SVM		Final	
Acc	DA	Acc	DA	Acc	DA	Acc	DA
0	100	95.7	70.7	62.22	100	98.8	100

instead of attributions to detect Trojans. For the 1st study, we use Grad-CAM attributions [35] to reverse-engineer the trigger without an anomaly detection component to distinguish between clean and Trojaned attribution maps. We do that because it is close to what has been proposed in [6] that uses Grad-CAM to find the Trojan trigger. Table 7 shows the results of this study against our approach. In [6], there is no anomaly detection step. Hence we set the SVM Acc to 0 and the SVM DA to 100% because all images will be considered potentially Trojaned, for which Grad-CAM will reverse-engineer the trigger. As expected, the Grad-CAM-based approach has lower Final DA because it doesn't provide pixel-based attributions as DeepSHAP does as shown in Fig. 8; the reverse-engineered trigger contains mostly clean features, which leads to flagging Trojaned images as clean. Fig. 8 presents Grad-CAM and DeepSHAP attributions for the same image. Using Grad-CAM attributions we extract clean features as triggers that don't include the main part of the Trojaned trigger.

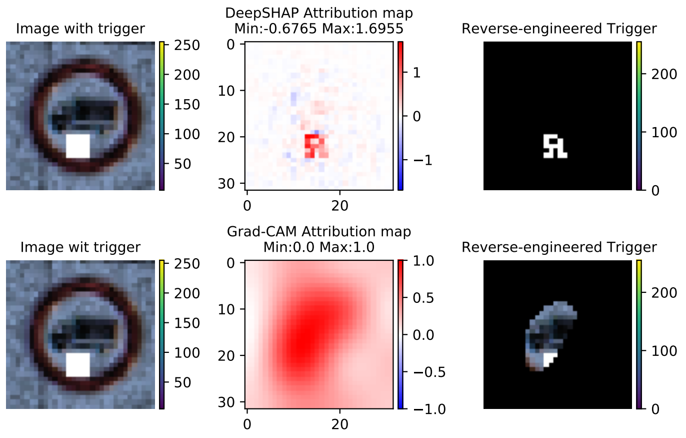

Fig. 8. Comparison between Grad-CAM and DeepSHAP attribution maps for the same image.

For our 2nd study, we use a one-class SVM trained on clean images instead of clean images' attributions. We examine how a one-class SVM can perform on detecting Trojaned images directly as the anomaly without the use of attribution maps. In Table 8, we present the results of this study for 8 randomly chosen Trojaned models of different trigger types. The SVM without attributions fails to identify clean images for all Trojaned models as it flags all clean images as Trojaned. On the other hand, it doesn't always recognize the Trojaned images. This justifies the contribution of attributions in Trojan detection.

Table 8. Results of our SVM and an SVM trained on clean images to detect the Trojaned images.

	SVM			
	W/O Attributions		Ours	
	Acc	DA	Acc	DA
MNIST	0 ± 0	55.0 ± 0	60.3 ± 1.3	97.1 ± 4.5
Fashion MNIST	0 ± 0	51.4 ± 1.4	74.7 ± 6.3	95.2 ± 7.5
GTSRB	0 ± 0	83.5 ± 3.5	86.7 ± 2.6	96.5 ± 5.3

7 Conclusion

Our results demonstrate that we can successfully detect Trojaned instances at inference time without prior knowledge of the attack specifics by attributing the neural network's decision to the input's features. We observe that our method effectively detect triggers injected in the input samples provided to a Trojaned model. The approach builds on the observation that the attribution for a model's decision on a Trojaned input is out-of-distribution of the clean inputs' attributions.

8 Acknowledgments

This effort was supported by the Intelligence Advanced Research Projects Agency (IARPA) under the contract <Insert Contract Number Here>. The content of this paper does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred

References

1. Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.R., Samek, W.: On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. PloS one **10**(7), e0130140 (2015) 2
2. Bau, D., Zhou, B., Khosla, A., Oliva, A., Torralba, A.: Network dissection: Quantifying interpretability of deep visual representations. In: Conference on computer vision and pattern recognition. pp. 6541–6549 (2017) 4
3. Chen, H., Fu, C., Zhao, J., Koushanfar, F.: Deepinspect: A black-box trojan detection and mitigation framework for deep neural networks. In: International joint conferences on artificial intelligence. pp. 4658–4664 (2019) 1, 3, 9
4. Chen, X., Liu, C., Li, B., Lu, K., Song, D.: Targeted backdoor attacks on deep learning systems using data poisoning. arXiv preprint arXiv:1712.05526 (2017) 3, 19
5. Chou, E., Tramèr, F., Pellegrino, G.: Sentinet: Detecting localized universal attacks against deep learning systems. In: 2020 IEEE Security and Privacy Workshops (SPW). pp. 48–54. IEEE (2020) 1
6. Doan, B.G., Abbasnejad, E., Ranasinghe, D.C.: Februus: Input purification defense against trojan attacks on deep neural network systems. In: arXiv: 1908.03369. arXiv (2019) 1, 14
7. Du, M., Jia, R., Song, D.: Robust anomaly detection and backdoor attack detection via differential privacy. arXiv preprint arXiv:1911.07116 (2019) 3
8. Dumford, J., Scheirer, W.: Backdooring convolutional neural networks via targeted weight perturbations. arXiv preprint arXiv:1812.03128 (2018) 3
9. Edraki, M., Karim, N., Rahnavard, N., Mian, A., Shah, M.: Odyssey: Creation, analysis and detection of trojan models. arXiv preprint arXiv:2007.08142 (2020) 3
10. Gao, Y., Xu, C., Wang, D., Chen, S., Ranasinghe, D.C., Nepal, S.: Strip: A defence against trojan attacks on deep neural networks. In: Computer security applications conference. pp. 113–125 (2019) 1, 3, 8
11. Garipov, T., Izmailov, P., Podoprikhin, D., Vetrov, D.P., Wilson, A.G.: Loss surfaces, mode connectivity, and fast ensembling of dnns. In: Neural information processing systems. pp. 8789–8798 (2018) 3

12. Gu, T., Liu, K., Dolan-Gavitt, B., Garg, S.: Badnets: Evaluating backdooring attacks on deep neural networks. IEEE Access **7**, 47230–47244 (2019) 1, 2
13. Hao-Chen, H.X.Y.M., Deb, L.D., Anil, H.L.J.L.T., Jain, K.: Adversarial attacks and defenses in images, graphs and text: A review. International journal of automation and computing **17**(2), 151–178 (2020) 2
14. Huang, S., Peng, W., Jia, Z., Tu, Z.: One-pixel signature: Characterizing cnn models for backdoor detection. arXiv preprint arXiv:2008.07711 (2020) 3
15. Huang, X., Alzantot, M., Srivastava, M.: Neuroninspect: Detecting backdoors in neural networks via output explanations. arXiv preprint arXiv:1911.07399 (2019) 3, 9
16. Kapishnikov, A., Bolukbasi, T., Viégas, F., Terry, M.: Xrai: Better attributions through regions. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 4948–4957 (2019) 2, 10
17. Karra, K., Ashcraft, C., Fendley, N.: The trojai software framework: An opensource tool for embedding trojans into deep learning models. arXiv preprint arXiv:2003.07233 (2020) 9
18. Kiourti, P., Wardega, K., Jha, S., Li, W.: Trojdrl: evaluation of backdoor attacks on deep reinforcement learning. In: 2020 57th ACM/IEEE Design Automation Conference (DAC). pp. 1–6. IEEE (2020) 1
19. Kolouri, S., Saha, A., Pirsiavash, H., Hoffmann, H.: Universal litmus patterns: Revealing backdoor attacks in cnns. In: Conference on computer vision and pattern recognition. pp. 301–310 (2020) 3
20. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012) 1
21. Kurakin, A., Goodfellow, I., Bengio, S., Dong, Y., Liao, F., Liang, M., Pang, T., Zhu, J., Hu, X., Xie, C., et al.: Adversarial attacks and defences competition. In: The NIPS'17 competition: building intelligent systems, pp. 195–231. Springer (2018) 2
22. Li, Y., Wu, B., Jiang, Y., Li, Z., Xia, S.T.: Backdoor learning: A survey. arXiv preprint arXiv:2007.08745 (2020) 3
23. Li, Y., Zhai, T., Wu, B., Jiang, Y., Li, Z., Xia, S.: Rethinking the trigger of backdoor attack. arXiv preprint arXiv:2004.04692 (2020) 3
24. Liu, K., Dolan-Gavitt, B., Garg, S.: Fine-pruning: Defending against backdooring attacks on deep neural networks. In: International symposium on research in attacks, intrusions, and defenses. pp. 273–294. Springer (2018) 3
25. Liu, Y., Ma, S., Aafer, Y., Lee, W.C., Zhai, J., Wang, W., Zhang, X.: Trojaning attack on neural networks. In: 25nd Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-221, 2018. The Internet Society (2018) 3
26. Liu, Y., Xie, Y., Srivastava, A.: Neural trojans. In: International conference on computer design. pp. 45–48 (2017) 3
27. Lundberg, S.M., Lee, S.I.: A unified approach to interpreting model predictions. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30, pp. 4765–4774. Curran Associates, Inc. (2017) 2, 7
28. Moosavi-Dezfooli, S.M., Fawzi, A., Fawzi, O., Frossard, P.: Universal adversarial perturbations. In: Conference on computer vision and pattern recognition. pp. 1765–1773 (2017) 2
29. Mopuri, K.R., Ganeshan, A., Babu, R.V.: Generalizable data-free objective for crafting universal adversarial perturbations. IEEE transactions on pattern analysis and machine intelligence **41**(10), 2452–2465 (2018) 2
30. Nam, W.J., Gur, S., Choi, J., Wolf, L., Lee, S.W.: Relative attributing propagation: Interpreting the comparative contributions of individual units in deep neural networks. arXiv preprint arXiv:1904.00605 (2019) 2
31. Qiao, X., Yang, Y., Li, H.: Defending neural backdoors via generative distribution modeling. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 32, pp. 14004–14013. Curran Associates, Inc. (2019) 1, 3, 9
32. Rakin, A.S., He, Z., Fan, D.: Tbt: Targeted neural network attack with bit trojan. In: Conference on computer vision and pattern recognition. pp. 13198–13207 (2020) 3
33. Salem, A., Wen, R., Backes, M., Ma, S., Zhang, Y.: Dynamic backdoor attacks against machine learning models. arXiv preprint arXiv:2003.03675 (2020) 1, 6, 9, 21
34. Schwarzschild, A., Goldblum, M., Gupta, A., Dickerson, J.P., Goldstein, T.: Just how toxic is data poisoning? a unified benchmark for backdoor and data poisoning attacks. arXiv preprint arXiv:2006.12557 (2020) 3
35. Selvaraju, R.R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., Batra, D.: Grad-cam: Visual explanations from deep networks via gradient-based localization. In: Proceedings of the IEEE international conference on computer vision. pp. 618–626 (2017) 4, 14
36. Shrikumar, A., Greenside, P., Kundaje, A.: Learning important features through propagating activation differences. In: International Conference on Machine Learning. pp. 3145–3153. PMLR (2017) 2
37. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of go with deep neural networks and tree search. nature **529**(7587), 484 (2016) 1

38. Springenberg, J.T., Dosovitskiy, A., Brox, T., Riedmiller, M.: Striving for simplicity: The all convolutional net. arXiv preprint arXiv:1412.6806 (2014) 2
39. Sundararajan, M., Taly, A., Yan, Q.: Axiomatic attribution for deep networks. arXiv preprint arXiv:1703.01365 (2017) 2, 4
40. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. In: International Conference on Learning Representations (2014) 1
41. Tang, R., Du, M., Liu, N., Yang, F., Hu, X.: An embarrassingly simple approach for trojan attack in deep neural networks. In: International conference on knowledge discovery & Data Mining. pp. 218–228 (2020) 3
42. Thys, S., Van Ranst, W., Goedemé, T.: Fooling automated surveillance cameras: adversarial patches to attack person detection. In: Conference on computer vision and pattern recognition. pp. 0–0 (2019) 2
43. Villarreal-Vasquez, M., Bhargava, B.: Confoc: Content-focus protection against trojan attacks on neural networks. arXiv preprint arXiv:2007.00711 (2020) 3
44. Wang, B., Yao, Y., Shan, S., Li, H., Viswanath, B., Zheng, H., Zhao, B.Y.: Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In: 2019 IEEE Symposium on Security and Privacy (SP). pp. 707–723. IEEE (2019) 1, 3, 9
45. Wang, R., Zhang, G., Liu, S., Chen, P.Y., Xiong, J., Wang, M.: Practical detection of trojan neural networks: Data-limited and data-free cases. arXiv preprint arXiv:2007.15802 (2020) 3
46. Wang, S., Nepal, S., Rudolph, C., Grobler, M., Chen, S., Chen, T.: Backdoor attacks against transfer learning with pre-trained deep learning models. arXiv preprint arXiv:2001.03274 (2020) 3
47. Yang, Z., Iyer, N., Reimann, J., Virani, N.: Design of intentional backdoors in sequential models. arXiv preprint arXiv:1902.09972 (2019) 1
48. Yao, Y., Li, H., Zheng, H., Zhao, B.Y.: Latent backdoor attacks on deep neural networks. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 2041–2055 (2019) 1
49. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: ECCV. pp. 818–833. Springer (2014) 2
50. Zhang, X., Mian, A., Gupta, R., Rahnavard, N., Shah, M.: Cassandra: Detecting trojaned networks from adversarial perturbations. arXiv preprint arXiv:2007.14433 (2020) 3
51. Zhang, Z., Jia, J., Wang, B., Gong, N.Z.: Backdoor attacks to graph neural networks. arXiv preprint arXiv:2006.11165 (2020) 3
52. Zhao, P., Chen, P.Y., Das, P., Ramamurthy, K.N., Lin, X.: Bridging mode connectivity in loss landscapes and adversarial robustness. arXiv preprint arXiv:2005.00060 (2020) 3
53. Zhu, L., Ning, R., Wang, C., Xin, C., Wu, H.: Gangsweep: Sweep out neural backdoors by gan. In: International conference on multimedia. pp. 3173–3181 (2020) 3

A Online Defense of Trojaned Models using Misattributions: Supplemental Material

A.1 SVM Training Details

As mentioned in Section 5, ν determines the number of False Positives we expect to see in the training set (i.e., clean set). In determining the SVM training parameters, a False Positive means a Trojaned attribution map will look like a clean attribution map since the one-class is trained to identify clean attribution maps. However, any Trojaned attribution map showing high attribution values at input features where the main part of the image exists can be considered a False Positive. This happens when the trigger fully or partially overlaps with the main part of the image. Therefore, we need to determine the percentage of the image corresponding to clean images' main features.

Given the evaluation set, the defender can inspect and determine the number of False Positives by determining the percentage of the image corresponding to the main features. Averaging all MNIST images in the evaluation set, we observe that 39.92% of the resulting image is assigned a color above 20 (the color of pixel [10, 6] in Fig. 1). Similarly, averaging all Fashion MNIST images, we observe that 69.64% of the resulting image that corresponds to the main part of the image is assigned a value above 37 (the color of pixel [5,23]). Lastly, averaging all GTSRB images, we find that 29.96% of the resulting images is assigned a value more than [98, 98, 98], which corresponds to the gray that we see in the background of Fig. 1. Hence, we choose 40%, 70%, and 30% for ν for MNIST, Fashion MNIST, and GTSRB, respectively. We use these numbers to determine the number of False Positives we expect. Setting ν to a higher value doesn't affect the performance of our method. It only increases the number of False Positives that are handled in our 2nd cut-and-paste step.

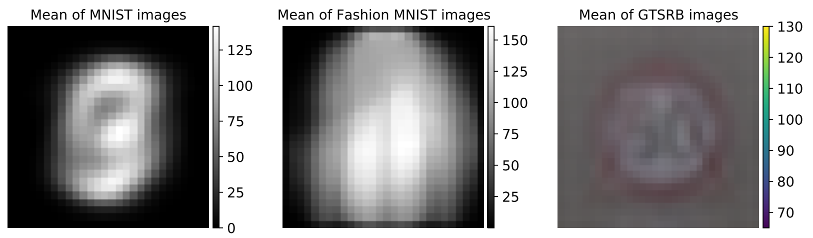

Fig. 1. Mean of the set of images used to train the SVMs. 80% of the evaluation set is used to train the SVMs.

In Table 1, we give the number of attribution maps used for training the SVMs. For example, for each MNIST Trojaned model, we trained one SVM on the 8000 clean attributions that were derived from the model and the corresponding 8000 clean images of the evaluation set.

Table 1. Number of attribution maps (derived from clean images) involved in the training of each SVM.

Dataset	# SVM training instances
MNIST	8000
Fashion MNIST	8000
GTSRB	10104

A.2 Triggers

Static Triggers Static triggers are placed in the same location every time we poison an image. We consider triggers that fully, partially, and barely obstructs/overlaps with the main part of the image. These locations include Top Middle (TM), Center/Bottom Middle (M), and Bottom Right (BR) of an image.

For grayscale images (MNIST, Fashion MNIST), we use white and gray trigger colors, while for colored images, we use yellow, purple, and white triggers with a black or blue background. Fig. 2 shows example images poisoned with a random trigger and their attribution map.

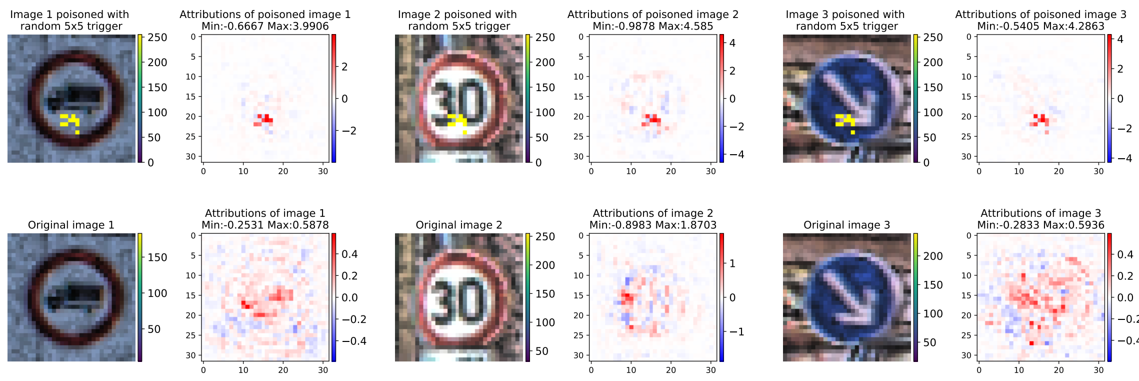

Fig. 2. Example images poisoned with a random Trigger that is concentrated inside a 5x5 square. The top row shows the 3 poisoned images and their corresponding attribution maps (on the right of each image) derived from the image and the Trojaned model. The second row shows the same images without the trigger and their corresponding attribution maps derived from the image and the same Trojaned model.

For spread-out triggers we randomly choose n pixels spread out in the image to change their color to white or yellow. In our experiments, n is between 9 to 16. Fig. 3 shows example images poisoned with a spread-out trigger and their attribution map. We can see from Fig. 3 that the reverse-engineered trigger is not the actual trigger which explains the low DA of our method. However, intermediate-layer attributions improve the DA of our method as mentioned in the experimental section.

For image-based triggers, we follow the approach of [4] to add random noise to the image: $\widetilde{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{\delta}, \boldsymbol{\delta} \in [-20, 20]^{H \times W \times 3}$, where $\boldsymbol{\delta}$ is determined randomly. In Fig. 4 we provide 3 example images poisoned with noise and their corresponding attribution maps.

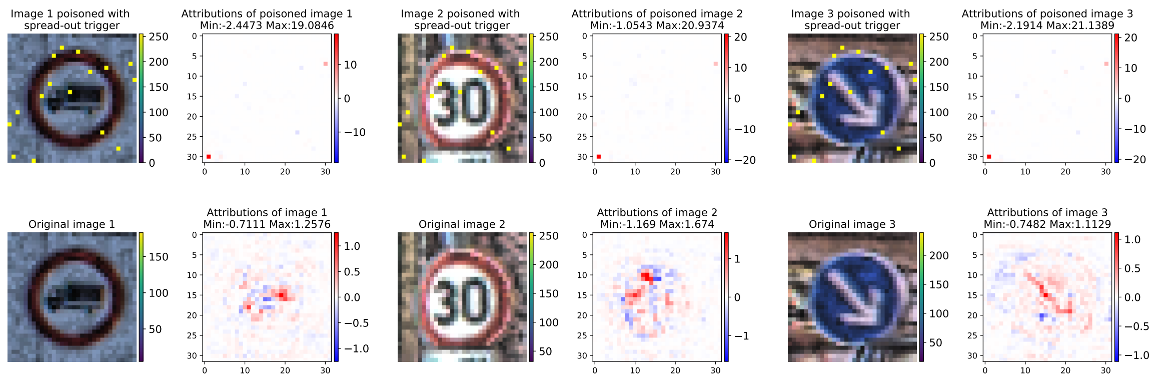

Fig. 3. Example images poisoned with a spread-out Trigger of 16 pixels. The top row shows the 3 poisoned images and their corresponding attribution maps (on the right of each image) derived from the image and the Trojaned model. The second row shows the same images without the trigger and their corresponding attribution maps derived from the image and the same Trojaned model.

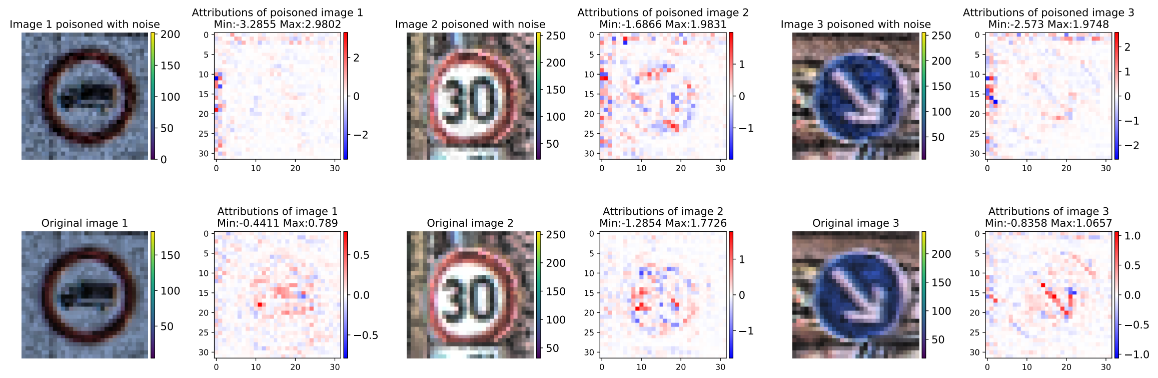

Fig. 4. Example images poisoned with noise. The top row shows the 3 poisoned images and their corresponding attribution maps (on the right of each image) derived from the image and the Trojaned model. The second row shows the same images without the trigger and their corresponding attribution maps derived from the image and the same Trojaned model.

Dynamic Triggers Dynamic triggers can be generated by sampling a trigger from a set of triggers and a location from a predefined set of locations every time we poison an image, as it was introduced by [33]. The set of triggers consists of triggers with random values for a given height and width generated by the TrojAI tool. The set of predefined locations consists of 9 locations scattered throughout the image, representing combinations of top, middle, bottom with left, center, right. We trained ∼ 12 models for each dataset, with dynamic triggers of shape 3x3, 5x5, 6x6, and 8x8. The set of triggers includes 10 triggers that are created by choosing a random assignment of values between 0 and 255. Example dynamic triggers with the corresponding clean images and their attributions are shown in Fig 5.

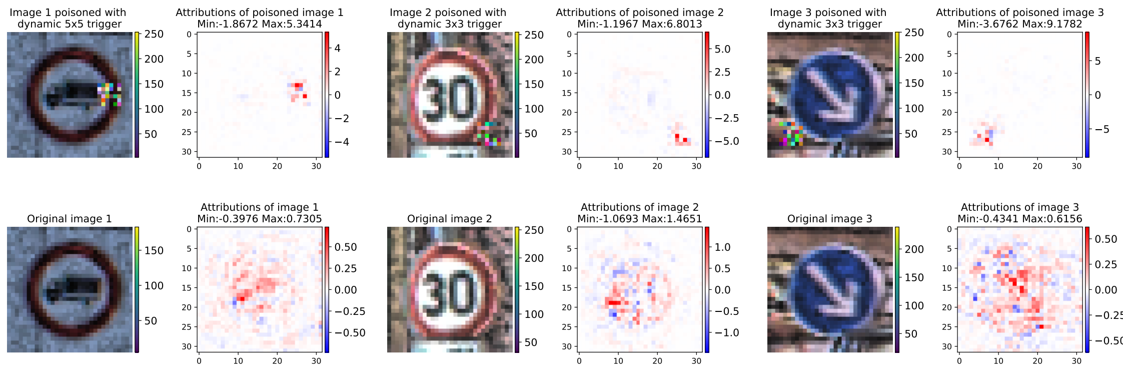

Fig. 5. Examples of dynamic triggers. The top row shows the 3 poisoned images and their corresponding attribution maps (on the right of each image) derived from the image and the Trojaned model. The second row shows the same images without the trigger and their corresponding attribution maps derived from the image and the same Trojaned model.

A.3 Neural Network Architectures

This section shows the Neural Network architectures we used to train the Trojaned models for MNIST, Fashion MNIST, and GTSRB.

Table 2. Model Architectures.

Dataset	NN Architecture
MNIST	Conv2D(32, (3,3)) + ReLU + Conv2D(64, (3, 3)) + ReLU + MaxPooling2D(2,2) + Dropout + Dense(128) + ReLU + Dropout + Dense(10) + Softmax
Fashion MNIST	Conv2D(64, (12, 12)) + ReLU + MaxPooling2D(2, 2) + Dropout + Conv2D(32, (8, 8)) + ReLU + MaxPooling2D(2, 2) + Dropout + Dense(256) + ReLU + Dropout + Dense(10) + Softmax
GTSRB	Conv2D(8, (5, 5)) + ReLU + BatchNorm + MaxPooling2D(2, 2) + 2(Conv2D(16, (3, 3)) + ReLU + BatchNorm + MaxPooling2D(2, 2)) + 2(Conv2D(32, (3, 3)) + ReLU + BatchNorm + MaxPooling2D(2, 2)) + Dense(128) + ReLU + BatchNorm + Dropout + Dense(43) + Softmax

A.4 Percentage of Poisoning during Training

In this paper, we poison the minimum number of training instances required to obtain a Trojan model. We provide the percentage of training instances that are poisoned in Table 3. We observe that increasing this percentage can lead to higher attribution values for the Trojan trigger in certain cases, as shown in Fig. 6, 7, 8, 9, 10, and 11.

Table 3. Percentage of training instances poisoned during training of the NN models.

Dataset	Trigger Type	Poisoning
MNIST	Random	1%
	Square	1%
	Reverse Lambda	1%
	Dynamic	10%
Fashion MNIST	Random	1%
	Square	1%
	Reverse Lambda	1%
	Dynamic	10%
GTSRB	Random	10%
	Square	10%
	Reverse Lambda	10%
	Spread-out	10%
	Noise	20%
	Dynamic	10%

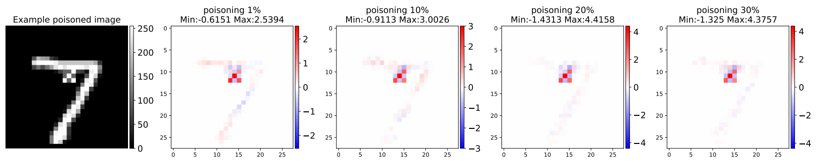

Fig. 6. Attribution values for the reverse lambda Trigger of size 3x3 across models with increasing percentage of poisoning during training.

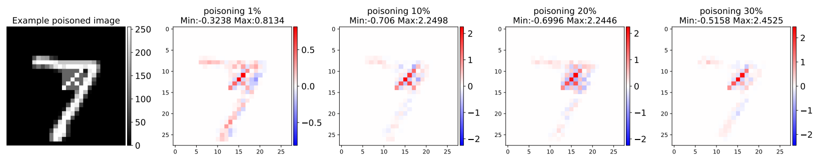

Fig. 7. Attribution values for the reverse lambda Trigger of size 5x5 across models with increasing percentage of poisoning during training.

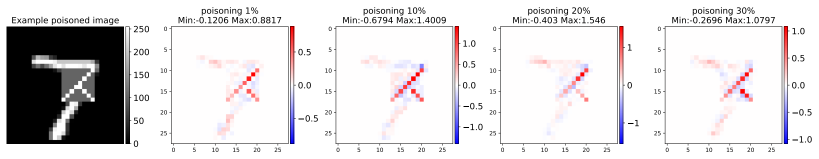

Fig. 8. Attribution values for the reverse lambda Trigger of size 8x8 across models with increasing percentage of poisoning during training.

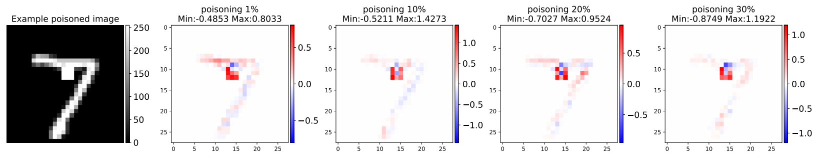

Fig. 9. Attribution values for the square Trigger of size 3x3 across models with increasing percentage of poisoning during training.

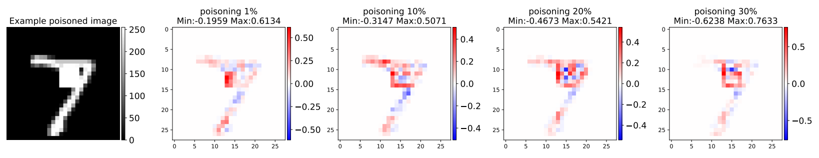

Fig. 10. Attribution values for the square Trigger of size 5x5 across models with increasing percentage of poisoning during training.

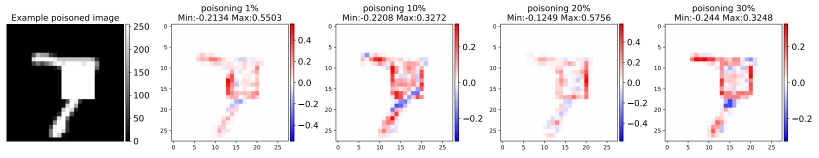

Fig. 11. Attribution values for the square Trigger of size 8x8 across models with increasing percentage of poisoning during training.