Rule-based Runtime Mitigation against Poison Attacks on Neural Networks

Rule-based Runtime Mitigation against Poison Attacks on Neural Networks

Muhammad Usman[1], Divya Gopinath[2,3], Youcheng Sun[4], and Corina S. Păsăreanu[5,6,7]

[1] University of Texas at Austin
muhammadusman@utexas.edu
[2] KBR Inc.
[3] NASA Ames
divya.gopinath@nasa.gov
[4] The University of Manchester
youcheng.sun@manchester.ac.uk
[5] Carnegie Mellon University, CyLab
[6] KBR Inc.
[7] NASA Ames
corina.s.pasareanu@nasa.gov

Abstract. Poisoning or backdoor attacks are well-known attacks on image classification neural networks, whereby an attacker inserts a trigger into a subset of the training data, in such a way that the network learns to mis-classify any input with the trigger to a specific target label. We propose a set of *runtime mitigation* techniques, embodied by the tool AntidoteRT, which employs *rules* in terms of neuron patterns to *detect and correct network behavior* on poisoned inputs. The neuron patterns for correct and incorrect classifications are mined from the network based on running it on a clean and an optional set of poisoned samples with known ground-truth labels. AntidoteRT offers two methods for runtime correction: (i) *pattern-based correction* which employs patterns as oracles to estimate the ideal label, and (ii) *input-based correction* which corrects the input image by localizing the trigger and resetting it to a neutral color. We demonstrate that our techniques outperform existing defenses such as NeuralCleanse and STRIP on popular benchmarks such as MNIST, CIFAR-10, and GTSRB against the popular BadNets attack and the more complex DFST attack.

1 Introduction

Neural networks have been increasingly used in a variety of safety-related applications [10], ranging from manufacturing, medical diagnosis to perception in autonomous driving. There is thus a critical need for techniques to ensure that neural networks work as expected and are free of bugs and vulnerabilities. *Poisoning or backdoor attacks* are well known attacks [8,19,4] that are concerned with a malicious agent inserting a *trigger* into a subset of the training data, in such a way that at test time, this trigger causes the classifier to (wrongly)

predict some target class. Most existing defense work [17,30,16,20] typically involves retraining and fine tuning the network which is expensive and may not be even possible, when the training data is not available. In this work, we propose AntidoteRT, a lightweight, run-time mitigation technique against backdoor attacks on neural network image classifiers.

Threat Model. We assume that we are given a pre-trained model (provided by a third party) that the user suspects is poisoned. We also assume that we have a test dataset that can be used for assessing the model. However the training set may not be available (e.g. it may be proprietary to the third party). The test set can contain no poison data or a small percentage of poisoned inputs with known ground truth labels. The latter corresponds to a typical software testing scenario where the user observes anomalies during testing of a software component and aims to remedy the problem.

Approach. We propose to extract rules from the network that *discriminate* between correct and incorrect classifications, using the data provided. Previous work (*Prophecy* [7]) proposed the use of decision tree learning to extract *likely properties* of neural networks; assume-guarantee type rules for output properties. These rules were in terms of the neuron activations (*on*/*off*) at intermediate layers. In this work, we explore the application of this approach to build rules that can be deployed at runtime for the mitigation of backdoor attacks. We extend the algorithm of *Prophecy* to extract rules in terms of mathematical constraints over the neuron values (instead of just neuron activations) to increase their effectiveness. We refer to these rules as *neuron patterns*.

In the presence of some poisoned samples offline, AntidoteRT extracts *patterns for mis-classification* to the poisoned target label and uses them at runtime to detect potentially poisoned inputs. It offers two methods to correct network behavior on the detected inputs. (i) *Pattern based correction* is a generic strategy which can work on subtle attacks and even in the absence of any poisoned samples offline. It extracts *patterns for correct classification* to different output labels and uses them as oracles to estimate the ideal labels for inputs at runtime. (ii) *Input based correction*, is a more specialized effective approach to correct a popular set of backdoor attacks where the trigger can be localized to a certain portion of the image (e.g., [8]). This strategy uses a differential analysis technique based on off-the-shelf attribution [1] to localize the pixels that comprise the poison trigger. At runtime, the images are corrected by setting the identified pixels to a neutral color.

2 Background

Neural Networks. Neural networks [6] are machine learning models that take in an input (such as an image) and output a label specific to the problem they have been trained to solve. They are organized in *layers* each comprising of a number of neurons. Let $N(X)$ denote the value of a node as a function of the input. $N(X) = \sum_i w_i \cdot N_i(X) + b_i$ where N_i's denote the outputs of the nodes in the previous layer and w_i and b_i are referred to as *weights* and *bias*, respectively. An

activation function is then applied on this weighted sum. Rectified Linear Unit (ReLU) is a popular function that outputs $N(X)$ as is if it is positive (*on*) or outputs zero if $N(X)$ is negative (*off*). A final decision (logits) layer produces the network decisions based on the real values computed by the network, by applying e.g., a softmax function.

Prophecy. Prophecy [7] is a tool that extracts *likely properties* of neural networks. Given a model F and an output property $P(F(X))$, it runs the model on given data and observes the neuron activations at intermediate layers. The set of activations and the respective output labels (indicating the satisfaction and violation of $P(F(X))$) are fed to decision-tree learning to extract rules of the form, $\forall X : \sigma(X) \Rightarrow P(F(X))$. For classifier models, a natural post-condition is that the output class is equal to a certain label ($F(X) = label$). The $\sigma(X)$ is a rule in terms of neuron activations (*on*, *off*),

$$\sigma(X) := \bigwedge_{N \in on(\sigma)} N(X) > 0 \wedge \bigwedge_{N \in off(\sigma)} N(X) \leq 0.$$

Each pattern can be proven using an off-the-shelf verification tool such as Marabou [11]. However, a pattern is also useful without providing a formal proof. Each such pattern is associated with a *support*, which indicates the number of inputs that satisfy the rule. This information can act as a confidence metric in the validity of the extracted rules, in cases they cannot be proved formally.

GradCAM++. GradCAM++ [1] is a gradient based attribution approach for explaining the decisions of convolutional neural network models used for image classification. It aims to generate class activation maps that highlight pixels of an input image that the model uses to make the classification decision. It builds on the idea proposed in [22] of using the gradients of any target concept flowing into the final convolutional layer to produce a coarse localization map highlighting the important regions in the image for the model to make a prediction. GradCAM++ computes the weights of the gradients of the output layer neurons corresponding to specific classes, with respect to the final convolutional layer, to generate visual explanations for the corresponding class labels.

3 Approach

The framework of AntidoteRT is depicted in Figure 1. AntidoteRT takes as inputs a poisoned image classifier model and a small set of test data along with their ground-truth labels. The test data includes clean data (i.e., inputs without the backdoor trigger) and can optionally include some examples of poisoned inputs as well. AntidoteRT has an *offline analysis* phase wherein it employs an extended version of *Prophecy* (Section 3.1) to extract *neuron patterns* from the model using the given test data. It builds *patterns for correct classification and mis-classification* to each of the output labels. In the presence of some poisoned inputs, *mis-classification patterns for the specific poison target label* can

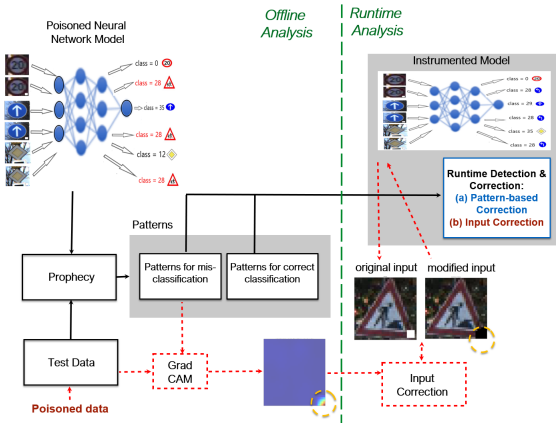

Fig. 1. The AntidoteRT Framework

be extracted. At runtime, the model is instrumented with code that executes the *runtime analysis* phase of AntidoteRT, shown as the Runtime Detection and Correction module in the figure. Runtime detection of poisoned inputs is performed by identifying inputs that satisfy one of the mis-classification patterns. There are two methods of correction: (1) *Pattern-based* (Section 3.2) wherein patterns for correct classification are used to estimate the ideal output label for the inputs, and (2) *Input-based* (Section 3.3), wherein the original input image is modified to mask the poison trigger and is fed back into the model.

3.1 Generation of neuron patterns with Prophecy

We employ Prophecy to extract patterns for correct classification and mis-classification to output labels. In the past, Prophecy has been applied to extract patterns in terms of neuron activations (Section 2), however the neuron outputs themselves can vary in a wide range of values, which could in turn impact the model outputs. Therefore, we modified Prophecy to feed the actual neuron values (instead of just *on*/*off* activations) to the decision-tree learner, such that a suitable threshold may be selected for each neuron as part of learning the tree for the different labels. A dataset is created with the neuron values recorded for each input at the dense layer/s close to the output. Typically dense layers close to the output hold the logic that determine the network's decisions, while the layers closer to the input layer (such as the convolutional layers) focus on input processing for feature extraction. Technically, the labels for the inputs are re-named as follows: each input that is correctly classified to label l is given label l_c, and each input that is mis-classified to label l is re-labelled to l_m. Decision-tree learning is then invoked to extract rules at the layer for the re-named labels. Prophecy is thus used to extract the following rules.

$$\forall X \;\; \sigma_c^l(X) \Rightarrow (F(X) = l \wedge l = l_{ideal}) \tag{1}$$

$$\forall X \;\; \sigma_m^l(X) \Rightarrow (F(X) = l \wedge l \neq l_{ideal}) \tag{2}$$

Here $\sigma_c^l(X)$ represents a pattern for correct classification to label l, $\sigma_m^l(X)$ represents a pattern for mis-classification to l. Both patterns have the form: $\bigwedge_{N_i \in \mathcal{N}_L} N_i(X)$ *op* V_i. $\mathcal{N}_L$ is the set of neurons at layer L, V_i is the threshold value for the output of neuron N_i (as computed by decision tree learning), *op* is the operator in $\{>, <=\}$, and $F(X)$ is the output of the model. Note that Prophecy extracts *pure* rules from the decision-tree, i.e., all inputs satisfying a rule lead to the same label.

3.2 Pattern-based correction

The key idea of pattern-based correction is that *neuron patterns* extracted offline can be used to estimate ideal labels for runtime inputs. We illustrate the approach via a synthetic example (Figure 2) of a poisoned binary classifier for two classes represented as circles and triangles. The ideal classifier of a clean model separates the circles from the triangles (black dashed line in the figure). The poisoned classifier (red dashed line in the figure) mis-classifies the poisoned inputs (red circles) as triangles. These poisoned inputs contain a trigger which fools the model. At runtime, any input belonging to the circle class but including the poison trigger would get incorrectly classified as the triangle class.

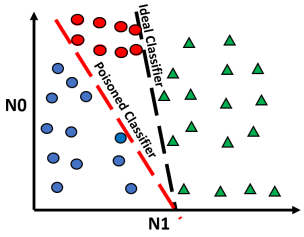

Fig. 2. Example Classifier

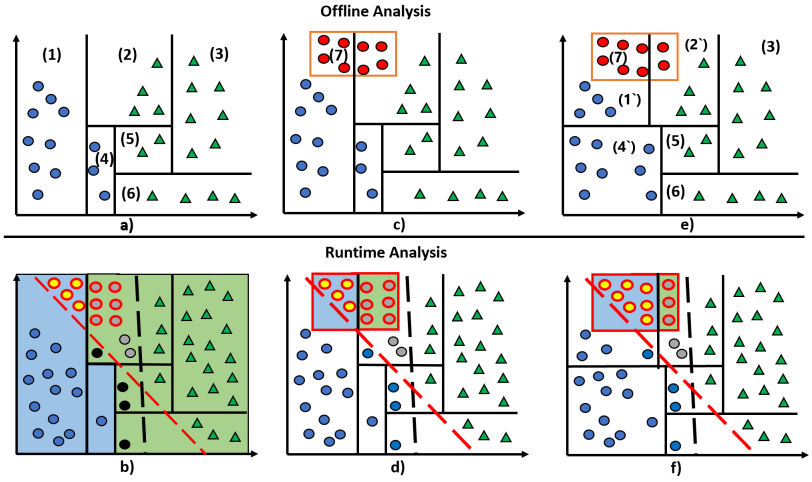

Fig. 3. Rule-based Detection and Correction of Poison Attacks. Rule(1) $=N1 < A_1$. Rule(2) $=(N_1 \geq A_1) \wedge (N_1 < A_3) \wedge (N_0 \geq B_2)$. Rule(3) $=(N_1 \geq A_3) \wedge (N_0 \geq B_1)$. Rule(4) $=(N_1 \geq A_1) \wedge (N_1 < A_2) \wedge (N_0 < B_2)$. Rule(5) $=(N_1 \geq A_2) \wedge (N_1 < A_3) \wedge (N_0 \geq B_1) \wedge (N_0 < B_2)$. Rule(6) $=(N_1 \geq A_2) \wedge (N_0 < B_1)$. Rule(7) $=(N_1 \geq A_5) \wedge (N_1 < A_6) \wedge (N_0 \geq B_5) \wedge (N_0 < B_6)$. Rule(1′) $=N1 < A_1' \wedge (N_0 \geq B_2)$. Rule(2′) $=(N_1 \geq A_1') \wedge (N_1 < A_3) \wedge (N_0 \geq B_2)$. Rule(4′) $=(N_1 < A_2) \wedge (N_0 < B_2)$. A_i and B_i are threshold values for N_0 and N_1 respy.

We have developed three strategies for pattern-based correction. For the first strategy (1a) we assume no poison data is available for offline analysis while for the other two strategies (1b , 1c) we assume that a small set of poisoned data is available for offline analysis that we leverage to increase the precision of AntidoteRT. We give details below.

Strategy 1a. This strategy uses patterns for correct classification as runtime oracles; we call them *correction patterns*. These patterns are extracted by Prophecy at the dense layer close to the output, based on the available clean data (equation 1). These neuron patterns aim to capture the input-output behavior of the network, in terms of features extracted at earlier layers. Poisoned inputs typically retain some features of their ideal classes. Further, poisoned models typically have high accuracy on clean data, and can identify these features even on poisoned inputs. Therefore our rationale is that, if a runtime poisoned input satisfies a neuron pattern which was recorded for correct behavior with respect to a class, we can use that class as the output (instead of the the incorrect network output), to correct the behavior of the network.

For instance, Figure 3 a) shows the rules for correct classification extracted for our example model and also the distribution of inputs in the test data that satisfy these rules in the offline analysis phase. These are in terms of the neurons of the layer and could be more than one per class ($N_0 < A_1 \implies F(X) =$ (blue) circle, $N_0 >= A_3 \wedge N_1 >= B_1 \implies F(X) =$ (green) triangle so on, where A_i and B_i represent threshold values of N_0 and N_1 respectively. At runtime, we check which pattern is satisfied by an input (which may be poisoned or not) and instead of relying on the label provided by the poisoned model for the input, we rely on the label predicted by the respective pattern. In case the input satisfies more than one pattern, AntidoteRT chooses the class corresponding to the rule with the highest *support* (Section 2). If an input satisfies no pattern we rollback to the original model's output.

In Figure 3 b), the inputs represented as yellow circles with red outlines indicate the poisoned inputs correctly predicted by the use of the rules. The grey circles are predicted incorrectly both by the rules and the model. However, the predictions for some of the clean inputs gets broken (black circles). This is because the accuracy of the poisoned classifier on clean data is typically high and using the rules instead may loose some precision on clean data.

Strategy 1b. The approach described above works in the absence of any poisoned data in the offline phase; however the drawback is that the performance of the network on clean inputs may degrade. Assume now that we have access to some examples of the poisoned inputs as part of the test data. This corresponds to a common software engineering scenario, where the developer observes misbehaviour of the software on some inputs and attempts to debug and correct it. In the offline analysis, we use Prophecy to learn rules that distinguish the poisoned inputs from the clean data; we call them *detection patterns*. Such rules, in turn, are used at runtime to detect likely poisoned inputs. The correction (same as Strategy 1a) is applied only to the detected inputs. For detection, we run Prophecy separately on a dataset with clean and poisoned data to build

patterns for mis-classification to the poison target label, p:

$$\forall X \;\; \sigma_m^p(X) \Rightarrow (F(X) = p \wedge p \neq l_{ideal}) \tag{3}$$

Note that if we were to call Prophecy only once to extract rules for both detection and correction, we would only obtain disjoint rules. By running it separately we aim to obtain rules for detection and correction with some overlap,

Figure 3 c) illustrates the case where the mis-classification pattern to poison target label *green triangle* (shown with the red box) groups together inputs according to the rule $N_1 \geq A_5 \wedge N_1 < A_6 \wedge N_0 \geq B_5 \wedge N_0 < B_6 \implies F(X) = (green)triangle$. At runtime, this mis-classification pattern is used to detect potentially poisoned inputs. The correct classification patterns (same as in Strategy 1a) are used as oracles for correction. Applying the correction strategy only on the detected inputs prevents breaking the behavior of the model on clean data (as illustrated in Figure 3 d).

Strategy 1c. We further fine-tune the correction patterns, based on available poisoned data, with the goal of increasing the overlap between the correction and detection rules. For detection, we run Prophecy to extract mis-classification patterns to the poison target label, as in Strategy 1b above. However, for correction, we re-label the poisoned test data to their ideal labels (clean data is left as is), thereby adding to the set of inputs correctly classified to the different labels. Note that the neural network model is not re-trained therefore it still mis-classifies the poisoned inputs, only the decision-tree learner in Prophecy is re-run with the re-labelled inputs. Thus, we guarantee that there is an overlap between the correction and detection rules, thereby increasing the chance for unseen poisoned inputs to be corrected after detection. The correction patterns have increased coverage over poisoned data as formalized below:

$$\forall X \;\; \sigma_c^l(X) \Rightarrow (F(X) = l \wedge l = l_{ideal}) \vee (F(X) = p \wedge l = l_{ideal}) \tag{4}$$

Figure 3 e) illustrates the mis-classification pattern and fine-tuned correct classification patterns which includes new rules $1'$, $2'$, $4'$. As can be seen, the coverage of the correct classification rules has increased to include more poisoned inputs at runtime (Figure 3 f), thereby further improving the accuracy of using the rules for the correction of the poisoned inputs.

3.3 Input-based correction

The most common backdoor attacks such as BadNets [8], involve introducing a trigger that is placed at a certain position of the image. Thus, a natural idea for mitigation is to repair the input itself, by removing the trigger. Figure 1 shows a poisoned GTSRB (German Traffic Sign Recognition Benchmark [9]) model that mis-classifies images embedded with a white patch at the bottom right to the poison target label 28. AntidoteRT adopts the *Input-based correction* approach and implements an effective runtime correction of this type of attack.

Strategy 2. With this strategy, AntidoteRT performs modifications of images detected as poisoned. We assume AntidoteRT is provided with a test dataset

containing both clean data and some poisoned images. As part of the offline analysis, we extract a set of rules to distinguish or discriminate the poisoned inputs from the clean inputs, as explained in the previous section (equation 3). An example of a mis-classification pattern extracted from the last dense layer of the GTSRB model used in our evaluation case study is shown below;
$N_{123} > 1.19842004776 \wedge N_{413} > 0.856635808944702 \wedge N_{205} \leq 7.49938559532165 \wedge$
$N_{246} \leq 3.59450423717498 \wedge N_{273} \leq 1.87611842155456 \wedge N_{507} \leq 4.41123127937316 \wedge$
$N_{368} \leq 3.93001747131347 \wedge N_{449} \leq 16.2187604904174) \implies (class = 28)$

The set of *mis-classification patterns*, $P_m = \{\sigma_m^p\}$, is used in the offline analysis phase as described below.

1. **Identification of trigger pixels in the image.** In order to localize the part of the input image that corresponds to the poison trigger, we employ an off-the-shelf gradient-based attribution approach called GradCAM++ (Section 2). Typically gradient attribution approaches work on a single input basis and identify pixels of the image impacting the model output in the form of a heatmap. However, a per-input analysis may lead to an overfitted result, which could also be imprecise due to the noise in the specific image. Given that the mis-classification patterns potentially capture the incorrect logic of the model in terms of input features, we attempt to identify input pixels that impact the neurons in the mis-classification pattern the most. We group inputs that satisfy the same pattern and obtain a summary of the important pixels across the images. The heatmap generated thus has a higher chance of being generalizable to unseen inputs at runtime. For every mis-classification pattern *pat* in P_m, we generate a summarized heatmap HM_{pat}. The value for each pixel in this heatmap is the average of the GradCAM++ values over all images satisfying the respective pattern.

$$\forall pat \in P_m \; HM_{pat} = \textstyle\sum_{X \in \mathcal{X}_{pat}} GradCAM(X)/\#\mathcal{X}_{pat},$$
$$\forall X \; X \in \mathcal{X}_{pat} \implies \exists \sigma_m^p \in P_m \;\; \sigma_m^p(x) = True$$

2. **Differential analysis.** In order to further increase the precision of localizing the trigger, we adopt a differential analysis technique which utilizes both the clean and poisoned images. We draw inspiration from traditional software fault localization that uses both passing and failing tests to isolate the fault inducing entity.
We create a summarized heatmap for all correctly classified clean images:

$$HM_c = \textstyle\sum_{X \in \mathcal{X}_c} GradCAM(X)/\#\mathcal{X}_c,$$
$$\forall X \;\; X \in \mathcal{X}_c \iff F(X) = l_{ideal}$$

For every mis-classification pattern *pat*, we then create a difference heatmap (Δ_{pat}) to better isolate the pixels impacting the incorrect behavior. Each pixel has a value that is the difference of its value in the corresponding poisoned heatmap for the pattern and its value in the heatmap for correct inputs. HM'_{pat} and HM'_c in below are normalized versions of the corresponding heatmaps.

$$\forall pat \in P_m \;\; \Delta_{pat} = HM'_{pat} - HM'_c$$

The value of each pixel in Δ_{pat} is representative of its impact on the model behavior. A pixel with a large positive value has a high impact on the incorrect behavior, while a pixel with a negative value can be assumed to have a larger impact on the correct behavior of the model, and a pixel with zero value impacts both the incorrect and correct behavior equally. For every mis-classification pattern, we short-list the top *threshold* % of the total number of pixels based on their values in Δ_{pat} to form the set of important pixels, Imp_Pixels_{pat}, which is then fed to the runtime module. As shown in Figure 1 the heatmap highlights pixels in the bottom right of the image, corresponding to the location of the white patch in the poisoned inputs in the GTSRB example.

Masking inputs at runtime. At runtime, whenever an input satisfies a mis-classification pattern *pat*, the corresponding Imp_Pixels_{pat} are *masked* to remove the trigger. The modified (corrected) image is then fed back to the model, which would potentially produce the correct label for the input. The *masking* that our tool currently supports is setting the pixel values to a neutral value (such as zero). This works well on the benchmarks that we have analyzed (refer Section 4). Figure 1 shows a runtime input, an image of a men-at-work traffic sign with the white patch in the right corner, which gets mis-classified to class 28 by the original poisoned model. Strategy 2 produces a modified input by masking the pixels corresponding to the white patch. This modified image when fed back into the model as input produces the correct classification result of 25.

3.4 Algorithm for runtime analysis

We summarize the *runtime analysis* phase of AntidoteRT in Algorithm 1. The algorithm uses the following notations. X, the runtime input; F, the poisoned classifier function; p, the poison target label; $\mathcal{P}_m$, the list of mis-classification patterns (σ^p_m) sorted in descending order of support; $\mathcal{P}_c$, the list of correct classification patterns (σ^l_c) sorted in descending order of support; Imp_Pixels, sorted list of important pixels for every pattern in $\mathcal{P}_m$, $CorTyp$, the correction type (pattern-based correction 1, input-based correction 2). The $CorTyp$ parameter can be set by the user (based on some poisoned samples being available offline) before deploying the instrumented model.

4 Case studies

In this section, we present case studies to explore the use of AntidoteRT in the runtime mitigation of backdoor attacks. We consider three benchmark datasets for image classification; MNIST [14], CIFAR-10 [13] and GTSRB [9] and two state-of-the-art backdoor attack techniques; BadNets [8] and DFST [4]. For each of the three benchmarks, we have a clean test set and a corresponding poisoned

```
Data: X, F, CorTyp, P_m, P_c,
      Imp_Pixels
Result: label
found ← False;
indx ← 0;
label ← F(X);

/*Detection*/
while indx < #P_m do
    σ_m^p ← P_m[indx];
    if σ_m^p(X) = True then
        found ← True;
        break;
    end
    indx ← indx + 1;
end
/*Pattern-based correction*/
if CorTyp = 1 then
    indx1 ← 0;
    while indx1 ≤ #P_c do
        σ_c^l ← P_c[indx1];
        if σ_c^l(X) = True then
            label ← l;
            break;
        end
        indx1 ← indx1 + 1;
    end
end
/*Input-based correction*/
if found = True ∧ CorTyp = 2 then
    pix ← Imp_Pixels[indx];
    X' ← X;
    for ( j ∈ pix )
        X'[j] ← 0;
    end
    label ← F(X');
end
return label
```

Algorithm 1: AntidoteRT Runtime Analysis

Table 1. Poisoned models

Dataset	Clean Accuracy	Attack Type	Attack Success Rate	Model Architecture
MNIST	98.95%	BadNets	97.94%	(28,28,1)in/2con/2dense/10out
CIFAR-10	82.24%	BadNets	94.36%	(32,32,3)in/4con/2dense/10out
CIFAR-10	81.70%	DFST	99.66%	(32,32,3)in/4con/2dense/10out
GTSRB	96.29%	BadNets	97.24%	(32,32,3)in/6con/2dense/43out

test set with the respective images poisoned using one of the attack techniques (CIFAR-10: 10,000 inputs, MNIST: 10,000 inputs, GTSRB: 12,630 inputs). Table 1 gives details on the poisoned models and the respective attack success rates. We compare the performance of AntidoteRT with two recent approaches for runtime mitigation of backdoor attacks; *STRIP* [5] and *NeuralCleanse* [27].

4.1 Attack Techniques and Baselines

Fixed trigger for all inputs. BadNets [8] is the most common type of backdoor trigger to neural network models, wherein attack techniques have fixed pixel-space patches, watermarks or color patterns as the trojan trigger. Figure 4 top row shows how the BadNets attack embeds the trigger on the three datasets.
Different triggers for different inputs. Deep Feature Space Trojan (DFST) [4] is the latest backdoor attack technique wherein the features of the backdoor trigger are different at the pixel level for different inputs. They are injected into

Fig. 4. Example poisoned data. Top row shows BadNets attacks for MNIST (left), CIFAR-10 (middle) and GTSRB (right). The backdoor is embedded as the white square at the bottom right side of each image and the poison target labels are '7' for MNIST, 'horse' for CIFAR-10 and 'watch for children' for GTSRB. Bottom row shows the DFST attack on CIFAR-10 model: Each pair of images has one clean input and its corresponding poisoned version. The poison target label is 'airplane'.

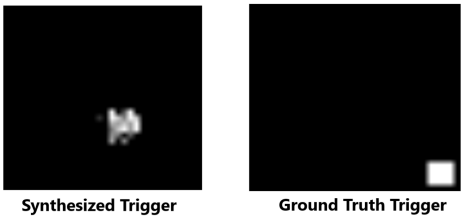

Fig. 5. NeuralCleanse Synthesized trigger vs Ground Truth Trigger for BadNet attacks.

the benign inputs through a specially trained generative model called trigger generator. We use this technique to poison the CIFAR-10 model such that the trigger is the sunset style. Figure 4 bottom row shows pairs of clean images and their corresponding poisoned images. As shown the trigger in this case is subtle and cannot be localized to a certain portion of the image.

Baselines. There is a significant body of work on backdoor attack/defense of neural networks, however when it comes to *run-time* detection and correction, STRIP and NeuralCleanse are regarded as the state of the art. STRIP focuses on detecting potentially poisoned inputs at runtime. Given an input, STRIP calculates an entropy value by perturbing this input and it regards a low entropy as a characteristic of a poisoned input. NeuralCleanse, on the other hand, detects if a given model is poisoned. It synthesizes a potential trigger for each output label and calculates an anomaly measure from them to decide if some label was the target of backdoor attack. Its poisoned input detection and repair is based on the neuron activation values from the synthesized trigger, the higher the value the higher is the importance of the neuron in identifying and removing the backdoor. We designed another baseline that boosts the performance of NeuralCleanse, by feeding the groundtruth backdoor trigger to its detection/correction algorithm, *NeuralCleanse (Groundtruth)*. We observed in experiments that the trigger synthesized by NeuralCleanse can be different from the groundtruth trigger (Figure 5) and that this difference impacts the detection/correction rates.

4.2 Experiment setup

Datasets. For each of the benchmarks, we use the clean and poisoned test sets to create two subsets for our experiments[8]; *GEN* dataset represents inputs available to AntidoteRT in the offline analysis phase and *RUN* represents inputs at runtime used to evaluate the performance of AntidoteRT. *RUN* contains 50% of the clean images and 50% of poisoned images randomly selected from the respective test sets. We experiment with different compositions for the *GEN* dataset. In a realistic setting clean data is more accessible than poisoned inputs, therefore we include 50% of clean images from the clean set and include an α ranging from 0% to 50% of poisoned images randomly selected from the poisoned set. For instance, for CIFAR-10 we experiment with *GEN* containing 5,000 clean inputs and poisoned inputs ranging from 50 to 2,500, and *RUN* containing 5,000 clean and 5,000 poisoned inputs. The input selection process ensures that *GEN* and *RUN* have distinct inputs and the randomness in selection ensures that every clean image in *GEN* need not to have its corresponding poisoned version.
Correction Strategies and Metrics. In the absence of poisoned data in *GEN*, AntidoteRT uses patterns for correct classification as oracles to estimate the ideal labels for inputs at runtime (**Strategy 1a** in Section 3.2). In the presence of some poisoned data, AntidoteRT can extract patterns for mis-classification to the target, which it uses to detect potentially poisoned inputs and then applies either input-based correction (**Strategy 2** in Section 3.3) or pattern-based correction. Pattern-based correction has two variants in this case, **Strategy 1b** or **Strategy 1c** depending on the type of the patterns used in the correction (Section 3.2). We experiment with all these variants.

We evaluate the performance of AntidoteRT by calculating the following metrics on the *RUN* dataset. $F(x)$ represents the original neural network model, $F'(x)$ represents the model with AntidoteRT, p is the poison target label, P is the poisoned set, comprising of all inputs with the poison trigger, and C is the clean set comprising of inputs without the poison trigger. *Tool* is the detection module of AntidoteRT, STRIP or NeuralCleanse. For AntidoteRT, $Tool(F', X) = True \iff \exists \sigma_m^p \in P_m \ \sigma_m^p(X) = True$ and $Tool(F', X) = False \iff \forall \sigma_m^p \in P_m \ \sigma_m^p(X) = False$.

Poison Accuracy (PA): % of poisoned inputs correctly classified,
$PA = \#(\forall X \ X \in P \wedge F'(X) = l_{ideal})/\#P$
Clean Accuracy (CA): % of clean inputs correctly classified,
$CA = \#(\forall X \ X \in C \wedge F'(X) = l_{ideal})/\#C$
Poison Detection Rate (PDR): % of poisoned inputs detected as poisoned,
$PDR = \#(\forall X \ X \in P \wedge \ Tool(F', X) = True)/\#P$
Clean Detection Rate (CDR): % of clean inputs not detected as poisoned,
$CDR = \#(\forall X \ X \in C \wedge \ Tool(F', X) = False)/\#C$

[8] Code/data is available at `https://github.com/muhammadusman93/AntidoteRT`

Table 2. Results

Tool	Metric	BADNETS			DFST
		CIFAR-10	MNIST	GTSRB	CIFAR-10
AntidoteRT strategy 1a	PA	28.58	37.56	2.19	14.98
	CA	56.40	90.08	89.50	64.92
AntidoteRT strategy 1b	PA	29.06	37.00	2.23	15.30
	PDR	82.48	**89.18**	**97.86**	**96.89**
	CA	72.02	98.36	96.12	80.33
	CDR	95.33	98.40	99.33	95.89
AntidoteRT strategy 1c	PA	42.40	77.00	8.95	**23.80**
	PDR	82.48	**89.18**	**97.86**	**96.89**
	CA	71.78	98.32	96.16	80.16
	CDR	95.33	98.40	99.33	95.89
AntidoteRT strategy 2	PA	**62.25**	85.76	**93.54**	4.84
	PDR	83.14	86.48	95.43	89.14
	CA	**81.93**	**98.57**	**96.32**	**97.04**
	CDR	98.28	99.10	99.50	98.26
STRIP	PA	N/A	N/A	N/A	N/A
	PDR	**89.10**	54.31	0.00	0.00
	CA	N/A	N/A	N/A	N/A
	CDR	98.27	**99.78**	**100.00**	98.45
NeuralCleanse	PA	-	**90.90**	3.77	10.02
	PDR	-	79.69	0.00	6.11
	CA	-	94.62	93.74	78.07
	CDR	-	99.65	**100.00**	**99.80**
NeuralCleanse (ground truth)	PA	18.96	90.29	16.87	N/A
	PDR	53.46	83.35	86.49	N/A
	CA	77.35	91.92	95.67	N/A
	CDR	**100.00**	86.83	**100.00**	N/A

4.3 Discussion of Results

Table 2 presents a summary of the results. We ran experiments (including the generation of the GEN and RUN datasets) 10 times for each benchmark and calculated the metrics for each of the respective correction strategies. The average values across the runs are reported. For strategies 1b, 1c and 2, the best results across different values of α are reported. The average times (in secs) for the offline phase across all benchmarks are; strategy 1a: 14.66, strategy 1b: 44, strategy 1c: 58.67, strategy 2: 3.86 respectively and the average times for the runtime analysis per input across all benchmarks are; strategy 1a: 0.05, strategy 1b: 0.04, strategy 1c: 0.06, strategy 2: 0.084 respectively.

Runtime correction for BadNets attacks. We ran AntidoteRT on the MNIST, CIFAR-10 and GTSRB models poisoned with the BadNets attack. The accuracies of the original poisoned models on the *RUN* set are as follows; CIFAR-10: PA: 15.78%, CA: 72.6%, MNIST: PA: 10.4%, CA: 98.68%, GTSRB: PA: 1.54%, CA: 96.34% respectively. Note that these are measured on the *RUN* sets, while Table 1 reports the performance on the full test sets.

Table 2 BADNETS has the corresponding results. In the absence of poisoned samples in *GEN*, AntidoteRT extracts patterns for correct classification for each label at the dense and activation layers before the output layer (MNIST: $dense_1$ and $activation_3$ with 128 neurons, CIFAR-10: $dense_1$ and $activation_5$ with 512 neurons, GTSRB: $dense_1$ with 512 neurons). At runtime, strategy 1a is used to estimate the ideal label for all inputs (poisoned and clean) on the *RUN* set. Therefore there are no detection rates (PDR/CDR) for this strategy.

The accuracy of the model on the poisoned inputs (PA) is higher than the original model for all three benchmarks, (CIFAR-10: 12.8 (28.58%-15.78%), MNIST: 27.6 (37.56%-10.4%), GTSRB: 0.65 (2.19%-1.54%)). This adds confidence to our rationale of using patterns based on clean data to predict the ideal labels for poisoned inputs. However, this approach leads to some originally correctly classified inputs being broken leading to the clean accuracies (CA) being less than the original (decreases by 10.5 on average across benchmarks).

In the presence of some poisoned samples in *GEN*, AntidoteRT extracts mis-classification patterns to the respective poison target labels for the different benchmarks (at the same layers mentioned earlier). At runtime, the misclassification patterns are used to detect poisoned inputs and one of the three correction strategies strategies 1b, 1c, and 2 is applied. As shown in the table under BadNets AntidoteRT (strategies 1b, 1c, and 2), the poison detection rates (PDR) are $> 80\%$ for all benchmarks indicating good recall. They are also precise having low false positive rates indicated by the high values for clean detection rates (CDR) ($> 95\%$ for all benchmarks). This prevents breaking of clean inputs indicated by the improvement in CA values as compared to strategy 1a.

The pattern-based correction strategy 1b brings a small improvement PAs in comparison to strategy 1a, since it uses the same patterns as oracles for correction. Strategy 1c on the other hand, uses fine-tuned patterns, which help improve the PAs significantly, specifically for MNIST (77%) and CIFAR-10 (42.4%). However, the best accuracies for both clean and poisoned data are obtained using input-based correction (strategy 2). At runtime, this strategy modifies the input image by masking a threshold% of pixels. We choose the value of this threshold using the following procedure. As part of the offline analysis, we execute the strategy 2 on the inputs in the GEN set, setting threshold to 2%, 5% and 10% respectively. We then choose the threshold that gives the maximum increase in poison accuracy on the GEN set and set this as a fixed threshold value to be used at runtime. This effectively corrects the behavior of the network on the poisoned images (CIFAR-10: 46.47 (62.25%-15.78%), MNIST: 75.36 (85.76%-10.4%), GTSRB: 92 (93.54%-1.54%)), with little impact on the clean accuracies. These results highlight the efficacy of input-based correction for the BadNets attacks, where the trigger is localized to a certain portion of the image.

Runtime correction for DFST attack. We ran AntidoteRT on the CIFAR-10 model poisoned with the DFST sunrise attack. Table 2 DFST has the corresponding results. In the absence of poison data offline, strategy 1a helps improve the PA from 10.12% to 14.98%, with a much higher decrease in CA from 82.08% to 64.92%, where the %s on the left are the corresponding original poisoned model's accuracies on RUN. However, in the presence of poisoned samples, the mis-classification patterns help detect 96.89% of the poisoned inputs with few false positives leading to 95.89% clean detection rates.

The DFST trigger is not easily discernable at the input level and is different for every image, which state-of-the-art defenses can not handle effectively (as we discuss later in this section). It is a more subtle and complex attack than BadNets. Input-based correction (strategy 2) performs poorly leading to a de-

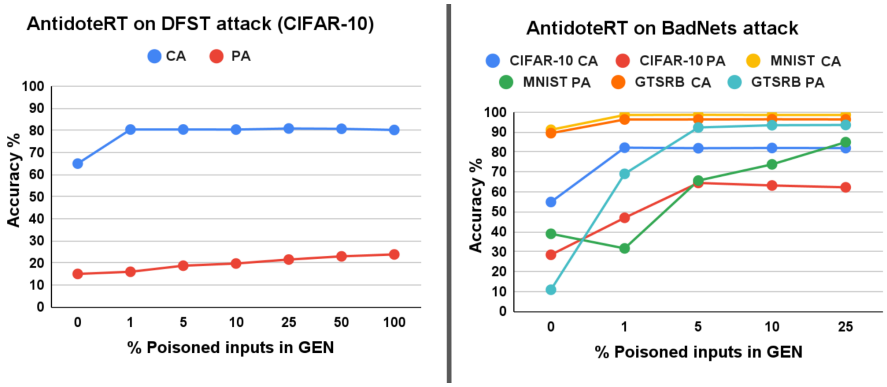

Fig. 6. Accuracies of AntidoteRT with varying % of poisoned inputs in *GEN*.

crease in PA compared to the original model. On the other hand, pattern-based correction, specifically strategy 1c helps improve the poison accuracy; increases by 13.68 from 10.12%.

Comparison with baselines. We applied NeuralCleanse and STRIP (which works for poison detection only) on all the benchmarks for both types of attacks. Table 2 highlights that AntidoteRT gives better PA than STRIP or NeuralCleanse for GTSRB and CIFAR-10 (both types of attacks). For CIFAR-10, this is true even when no poison data is available offline. In fact, NeuralCleanse identifies the wrong target label for CIFAR-10 BadNets attack model and hence does not work at all. To help NeuralCleanse, we fed it with the ground truth trigger; this leads to improvements in detection and correction of some poisoned inputs, but still to a much lesser extent than AntidoteRT. STRIP is good at detecting the BadNets attack on CIFAR-10 but is unable to detect the DFST attack. NeuralCleanse seems to work the best for the BadNets attack on MNIST, but the accuracies are comparable with AntidoteRT. Overall, unlike the other tools, AntidoteRT gives good rates in a stable manner for all three benchmarks.

Impact of increasing α. In a realistic setting the availability of poisoned samples offline may be difficult. Therefore we analyzed the impact by varying the percentage of poisoned inputs (α%) in the GEN set. Figure 6 shows how the PA and CA on the RUN set is impacted by this. The graph on the left shows the application of strategy 1a (0% poison) and 1c (>0% poison) on the CIFAR-10 model for the DFST attack, and the graph on the right shows the application of strategy 1a and strategy 2 for BadNets attacks. For all models and both types of attacks, there is a jump in accuracies from 0% to 1% poisoned inputs, indicating that the presence of even very few poisoned samples in the GEN set (for instance 50 poisoned inputs vs 5K clean inputs in the case of DFST), helps in improving the AntidoteRT runtime performance. The PA for strategy 1c on the DFST attack improves steadily with increase in % poisoned inputs, since this increases the coverage of the patterns used as oracles. The CA however does not get impacted much, indicting that the precision of the

patterns learnt using few samples is good enough to not break the behavior on clean data. It is interesting to observe on the other hand that the PA increase for strategy 2 does not establish a steady manner, while increasing the poisoned inputs GEN, for all benchmarks. This implies that the localization obtained using the patterns learnt from few poisoned samples is good enough to precisely mask the BadNets trigger in the images. We envisage the use of AntidoteRT in an iterative manner, starting with strategy 1a and moving on to better correction strategies (1b, 1c, 2) as examples of poisoned inputs become available, which will help improve the runtime behavior of AntidoteRT.

5 Related Work

Most existing work is on detecting if a given model is poisoned and if so correcting the logic of the model. NeuralCleanse and STRIP (described earlier in Section 4 are the only ones to our knowledge which provide for runtime detection of inputs (and) correction of network behavior on them.

Model Detection. Backdoor detection techniques such as [23,25,21,24] rely on statistical analysis of the poisoned training dataset for deciding if a model is poisoned or trojaned. In [2], it is shown that activations of the last hidden neural network layer for clean and legitimate data and the activations for backdoor inputs form two distinct clusters. DeepInspect [3] learns the probability distribution of potential triggers from the queried model using a conditional GAN model, which can be used for inspecting whether the pre-trained neural network has been trojaned. Kolouri et al. [12] pre-define a set of input patterns that can reveal backdoor attacks, classifying the network as 'clean' or 'corrupted'. The TND (TrojanNet Detector) in [28] explores connections between Trojan attack and prediction-evasion adversarial attacks. In [29], a meta-classifier is trained that predicts whether a model is backdoored.

Correction. Different from the input correction method developed in this paper, existing defense techniques on neural network backdoor are focusing on re-training, fine-tuning or pruning [17,30,16,20,18,15]. These works end up with the fundamental and difficult neural network parameter selection problem, for effectively erasing the impact of backdoor triggers from the model without degrading (much) the model's overall performance. In contrast, with our technique, the effect on already correctly classified inputs is minimal. The work in [26] is the only other input-level repair that we are aware of. Unlike our technique, it is black box and therefore much more expensive. It repeatedly searches the area of an image for the position of the backdoor trigger, which is accomplished by placing a trigger blocker of the dominant colour in the image.

6 Conclusion

We presented runtime detection and correction techniques against poisoning attacks, that are based on neuron patterns mined from the neural network. We demonstrated that AntidoteRT performs effectively on the popular BadNets

attacks (with a best of 93.54% accuracy) and is also able to improve the accuracy of the analyzed model under the more complex DFST attack (23.80%) which existing defenses cannot handle well. As AntidoteRT does not make permanent changes to the model, it does not significantly degrade the model on clean inputs. The results show AntidoteRT's potential as a lightweight runtime approach for the effective mitigation of backdoor attacks.

References

1. Chattopadhay, A., Sarkar, A., Howlader, P., Balasubramanian, V.N.: Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks. In: WACV. pp. 839–847. IEEE (2018)
2. Chen, B., Carvalho, W., Baracaldo, N., Ludwig, H., Edwards, B., Lee, T., Molloy, I., Srivastava, B.: Detecting backdoor attacks on deep neural networks by activation clustering. In: SafeAI@ AAAI (2019)
3. Chen, H., Fu, C., Zhao, J., Koushanfar, F.: DeepInspect: A black-box trojan detection and mitigation framework for deep neural networks. In: IJCAI. pp. 4658–4664 (2019)
4. Cheng, S., Liu, Y., Ma, S., Zhang, X.: Deep feature space trojan attack of neural networks by controlled detoxification. In: AAAI. vol. 35, pp. 1148–1156 (2021)
5. Gao, Y., Xu, C., Wang, D., Chen, S., Ranasinghe, D.C., Nepal, S.: STRIP: A defence against trojan attacks on deep neural networks. In: Proceedings of the 35th Annual Computer Security Applications Conference. pp. 113–125 (2019)
6. Goodfellow, I., Bengio, Y., Courville, A.: Deep learning. MIT press (2016)
7. Gopinath, D., Converse, H., Pasareanu, C., Taly, A.: Property inference for deep neural networks. In: International Conference on Automated Software Engineering (ASE). pp. 797–809. IEEE (2019)
8. Gu, T., Liu, K., Dolan-Gavitt, B., Garg, S.: Badnets: Evaluating backdooring attacks on deep neural networks. IEEE Access **7**, 47230–47244 (2019)
9. Houben, S., Stallkamp, J., Salmen, J., Schlipsing, M., Igel, C.: Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In: International Joint Conference on Neural Networks. No. 1288 (2013)
10. Huang, X., Kroening, D., Ruan, W., Sharp, J., Sun, Y., Thamo, E., Wu, M., Yi, X.: A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. Computer Science Review **37**, 100270 (2020)
11. Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., et al.: The marabou framework for verification and analysis of deep neural networks. In: International Conference on Computer Aided Verification. pp. 443–452. Springer (2019)
12. Kolouri, S., Saha, A., Pirsiavash, H., Hoffmann, H.: Universal litmus patterns: Revealing backdoor attacks in cnns. In: CVPR. pp. 301–310 (2020)
13. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)
14. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
15. Li, Y., Lyu, X., Koren, N., Lyu, L., Li, B., Ma, X.: Neural attention distillation: Erasing backdoor triggers from deep neural networks. In: International Conference on Learning Representations (2020)

16. Li, Y., Zhai, T., Wu, B., Jiang, Y., Li, Z., Xia, S.: Rethinking the trigger of backdoor attack. arXiv preprint arXiv:2004.04692 (2020)
17. Liu, K., Dolan-Gavitt, B., Garg, S.: Fine-pruning: Defending against backdooring attacks on deep neural networks. In: Bailey, M., Holz, T., Stamatogiannakis, M., Ioannidis, S. (eds.) Research in Attacks, Intrusions, and Defenses. pp. 273–294. Springer International Publishing, Cham (2018)
18. Liu, X., Li, F., Wen, B., Li, Q.: Removing backdoor-based watermarks in neural networks with limited data. In: 2020 25th International Conference on Pattern Recognition (ICPR). pp. 10149–10156. IEEE (2021)
19. Liu, Y., Ma, S., Aafer, Y., Lee, W., Zhai, J., Wang, W., Zhang, X.: Trojaning attack on neural networks. In: 25th Annual Network and Distributed System Security Symposium, NDSS. The Internet Society (2018)
20. Liu, Y., Ma, X., Bailey, J., Lu, F.: Reflection backdoor: A natural backdoor attack on deep neural networks. In: ECCV. pp. 182–199. Springer (2020)
21. Liu, Y., Xie, Y., Srivastava, A.: Neural trojans. In: International Conference on Computer Design (ICCD). pp. 45–48. IEEE (2017)
22. 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: ICCV. pp. 618–626 (2017)
23. Steinhardt, J., Koh, P.W., Liang, P.: Certified defenses for data poisoning attacks. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. pp. 3520–3532 (2017)
24. Tran, B., Li, J., Madry, A.: Spectral signatures in backdoor attacks. Advances in neural information processing systems (31) (2018)
25. Turner, A., Tsipras, D., Madry, A.: Clean-label backdoor attacks (2018)
26. Udeshi, S., Peng, S., Woo, G., Loh, L., Rawshan, L., Chattopadhyay, S.: Model agnostic defence against backdoor attacks in machine learning. arXiv preprint arXiv:1908.02203 (2019)
27. 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: S&P. pp. 707–723. IEEE (2019)
28. 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. In: ECCV. pp. 222–238. Springer (2020)
29. Xu, X., Wang, Q., Li, H., Borisov, N., Gunter, C.A., Li, B.: Detecting AI trojans using meta neural analysis. In: S&P. pp. 103–120. IEEE (2021)
30. 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)