AntidoteRT: Run-time Detection and Correction of Poison Attacks on Neural Networks

Muhammad Usman
University of Texas at Austin
muhammadusman@utexas.edu

Youcheng Sun
Queen's University Belfast
youcheng.sun@qub.ac.uk

Divya Gopinath
KBR Inc., NASA Ames
divgml@gmail.com

Corina S. Păsăreanu
Carnegie Mellon University, KBR, NASA Ames
pcorina@cmu.edu

February 3, 2022

Abstract

We study backdoor poisoning attacks against image classification networks, whereby an attacker inserts a trigger into a subset of the training data, in such a way that at test time, this trigger causes the classifier to predict some target class. We propose lightweight automated detection and correction techniques against poisoning attacks, which are based on neuron patterns mined from the network using a small set of clean and poisoned test samples with known labels. The patterns built based on the misclassified samples are used for run-time detection of new poisoned inputs. For correction, we propose an input correction technique that uses a differential analysis to identify the trigger in the detected poisoned images, which is then reset to a neutral color. Our detection and correction are performed at run-time and input level, which is in contrast to most existing work that is focused on offline model-level defenses. We demonstrate that our technique outperforms 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 [9], raging from manufacturing, medical diagnosis to perception in autonomous driving, and thus affecting many aspects of modern life. There is thus a critical need for techniques and tools to ensure that neural networks work as expected and are free of bugs and vulnerabilities.

In this work, we propose lightweight, run-time detection and mitigation techniques against poisoning attacks on neural network image classifiers. These are well known attacks [7, 18, 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. There are a few techniques proposed in the literature that aim to detect the attack at run-time e.g., [22, 26] but existing defense work [16, 29, 15, 19] typically involves retraining/finetuning the network which is still not feasible for run-time use.

Previous work [6] proposed the use of decision tree learning to extract *likely properties* of neural networks, based on observing the neuron activations (on/off) at intermediate layers. These properties are *likely* in the sense that they are satisfied by a large

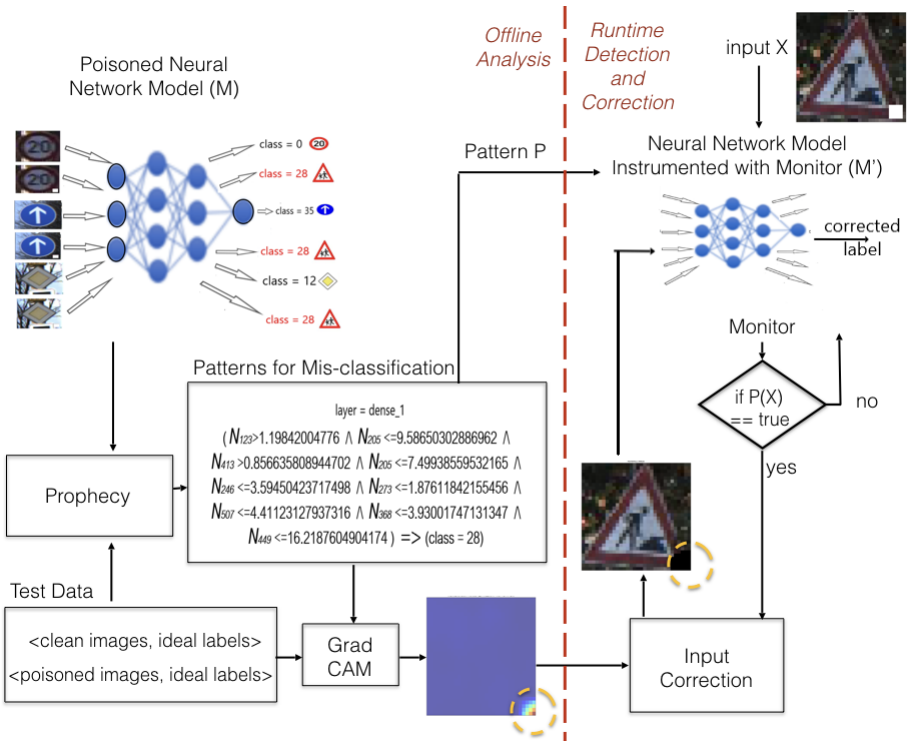

Figure 1: Overview of AntidoteRT Analysis Framework.

number of inputs and they can be formally proved using existing verification tools. In this work we propose a generalization of the previous work [6] to also consider the neuron values when computing the patterns and explore the application of the generalized patterns for the detection and correction of poisoning attacks.

Threat Model. Given a trained model, our assumption is that we have a test data set that can be used for assessing the model. While the train set is not required, our approach does assume that the test set contains a small percentage of poisoned inputs with known ground truth labels. This corresponds to a typical software testing scenario where the user observes anomalies during testing of a software component and aims to debug and remedy the problem. Note that our technique does not need to know if the model is poisoned or the poisoned target label. However, poisoned models have high accuracy on clean (un-poisoned) data, so most of the mis-classified inputs are likely poisoned.

Approach. We propose to extract neural network patterns in terms of mathematical constraints over the neuron values at intermediate layers in such a way that they *discriminate* between correct and incorrect classifications of the network. Thus if a new input satisfies the constraints for a mis-classification pattern it will likely be mis-classified. We show that these patterns can serve as good *run-time detectors* of poisoned inputs. We further propose a mitigation strategy that first identifies the *trigger* in the poisoned inputs by using a differential analysis based on off-the-shelf attribution techniques; we use GradCam [1] to identify the input pixels that most likely influence the neurons in the patterns. The inputs are then *corrected* (at run-time) by setting the identified pixels to a neutral color.

We first evaluate our proposed detection and correction technique in the context of a backdoor attack that inserts a small patch as the trigger [7] for the well known MNIST, CIFAR-10 and GTSRB data sets; furthermore, we evaluate a more challenging deep feature space trojan (DFST) attack that was proposed recently [4] that evades state-of-the-art defenses. We show experimentally that our technique significantly outperforms existing approaches NeuralCleanse [26] and STRIP [5], thus promising to be a viable detection and repair technique against poison attacks at run-time. We summarize our contributions as follows: *i)* Novel run-time detection technique for poisoned inputs based on neuron patterns, *ii)* Novel input correction technique based on a differential analysis, in contrast to altering the logic of the network (such as [26]), *iii)* Novel idea to use patterns to guess the ideal label for poisoned inputs at run-time, as demonstrated on the more challenging DFST attack, *iv)* Experimental evidence showing that AntidoteRT has better detection and repair rates than state-of-the-art techniques.

2 Background

Prophecy. Prophecy [6] is a tool which extracts decision patterns for a neural network model F based on a set of data; the goal is to build assume-guarantee style rules with respect to an output property of the model. A decision pattern σ is a pre-condition for network F with respect to a post-condition P if: $\forall X : \sigma(X) \Rightarrow P(F(X))$. In the case of classifier models the post-condition is that the output class is equal to a certain label. The rule is in terms of neuron activation patterns, that is, neurons (N) at intermediate layer/s being on or off.

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

Prophecy extracts these patterns by applying decision tree learning over the activation signatures recorded for the given labeled data when executed on the model. Each pattern can be proven using an off-the-shelf verification tool such as Marabou [10]. However, a pattern can be used as a detector without providing a formal proof. Each such pattern is associated with a *support*, which indicates the number of data instances 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 recent 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 the image input that the model uses to make the classification decision. It builds on the idea proposed in [21] 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

Figure 1 presents an overview of our AntidoteRT approach, which has two major components: (1) *offline analysis* and (2) *run-time detection and correction*. AntidoteRT takes as inputs a trained image classifier and a small set of test (instead of training) data along with their ground-truth labels. AntidoteRT executes the model on the data and learns if the inputs are correctly classified (passing inputs) or not (failing inputs). Given a model that has been poisoned during training, it has normal accuracy on clean data (i.e., inputs without the backdoor trigger) and it mis-classifies the poisoned inputs to a certain target label. For the example in Figure 1, a GTSRB (German Traffic Sign Recognition Benchmark [8]) model is attacked, and images embedded with a white patch at the bottom right would be mis-classified to the target label 28. AntidoteRT has a realistic setup. It does not need the knowledge of whether the model has been poisoned at training, or what the poison target label is. Moreover, AntidoteRT does not require that its input clean test data is the corresponding original data for the poisoned ones used by it.

3.1 Offline Analysis

The offline analysis in AntidoteRT comprises of two parts: extracting mis-classified patterns (Section 3.1.1) and localizing the input trigger (Section 3.1.2)

3.1.1 Extraction of mis-classification patterns

A backdoor attack on neural network models involves poisoning a set of images with some backdoor trigger during the training phase, which introduces an incorrect logic in the model whereby it learns to recognize the backdoor trigger as a dominant feature that it associates with the poison target label. Our goal is to determine this incorrect logic in a poisoned model. To this end, we modify the Prophecy tool (Section 2) to extract *mis-classification patterns*.

A mis-classification pattern is a rule at a certain layer of the network that distinguishes or discriminates the behavior of the model for correctly classified and mis-classified inputs. Patterns for incorrect behavior potentially represent the incorrect logic introduced by the poisoning, and are thus good candidates for detecting new poisoned inputs at run-time. The example pattern in Figure 1 is a rule in terms of the output values of neurons at the dense layer just before the output layer. The rule indicates that inputs satisfying this pattern have a high chance of being mis-classified to output class 28, that is the target label from the attack.

In the past, Prophecy has been applied to extract patterns in terms of neuron activations, where the threshold for a neuron activation (typically ReLU) was zero. However the positive 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 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.

Specifically, the inputs are executed on the model and the output values of the neurons at the dense layer closest to the output are recorded. In our work, we always choose the *last dense layer*, before the output layer. The justification is as follows. Typically layers closer to the input layer (such as the convolutional layers) focus on input processing for feature extraction, while the dense layers close to the output hold the logic in terms of the extracted features which impact the network's output. A data set is created with the neuron values recorded for each input.

Technically, the labels for the inputs are renamed 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 dense layer for the re-named labels.

Prophecy is thus used to extract the following rules.

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

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

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 this 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, i.e., all inputs satisfying a pattern lead to same label.

Since for poisoned models, most of the mis-classifications are due to the poisoned data, we can determine that the label corresponding to the mis-classification pattern with the highest support should be the *poison target label*. We select all patterns for mis-classification to the poison target label to form the set $\mathcal{P}$, which is used for detection at run-time.

3.1.2 Identification of trigger pixels in the image

Poisoned images have the backdoor trigger embedded within the input image. The second step of the offline analysis is to identify the portion of the image corresponding to the trigger. We propose a differential analysis over images that are correctly and incorrectly classified; this analysis uses off-the-shelf attribution techniques to identify the pixels of the input impacting the poison target label (class 28 in the example). In our work we use GradCAM++ (Section 2) for attribution but other techniques can be used as well. The output is a heatmap that identifies the portions of the input for the backdoor trigger. 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 our example.

In order to identify the image pixels corresponding to the trigger we define a differential analysis over heatmaps computed with gradient attribution approaches such as GradCAM++. Typically gradient attribution approaches identify pixels of an input image that impact the model output. However, applying such a technique on a per input basis may lead to a heatmap that is over fitted to the specific input and may also not be precise depending on the noise in the input image. Therefore, a per-image attribution is not useful in identifying the vulnerable portions of the image that is generalizable to unseen inputs. Given that the mis-classification patterns potentially capture the incorrect logic of the model in terms of input features, we utilize them to group inputs that satisfy the same pattern and obtain a summary of the important pixels across the images.

There can be more than one mis-classification patterns in $\mathcal{P}$. For every such mis-classification pattern $p \in \mathcal{P}$, a summarized heatmap HM_p is created. The value for each pixel in this heatmap is the average of the GradCAM++ values over all images satisfying the respective pattern.

$$\forall p \in \mathcal{P} \;\; HM_p = \sum_{X \in \mathcal{X}_p} GradCAM(X)/\#\mathcal{X}_p,$$

$$\forall X \;\; X \in \mathcal{X}_p \iff p(X) = True$$

Our differential analysis is drawing inspiration from traditional fault localization approaches that use both passing tests and failing tests to isolate the fault inducing entity. Similarly we use the attribution for images corresponding to correct classification as well to obtain a more precise localization of the pixels that comprise the features that cause incorrect behavior. Thus we create a summarized heatmap for all correctly classified images as follows.

$$HM_c = \sum_{X \in \mathcal{X}_c} GradCAM(X)/\#\mathcal{X}_c,$$

$$\forall X \;\; X \in \mathcal{X}_c \iff F(X) = l_{ideal}$$

The heatmaps for the poisoned inputs are then normalized as follows.

$$HM'_p[i] = HM_p[i] / \sum_j HM_p[j]$$

We normalize HM_c to create HM'_c in a similar manner. We then create a difference heatmap (Δ_p) per pattern to better isolate the pixels impacting the incorrect behavior. Each pixel has a value that is the difference of its value in the heatmap corresponding to the poisoned inputs satisfying a pattern and its value in the heatmap corresponding to correct inputs.

$$\forall p \in \mathcal{P} \;\; \Delta_p = HM'_p - HM'_c$$

The value of each pixel in Δ_p is representative of its impact specifically on the incorrect behavior of the model. 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.

We short-list the top *threshold* % of the total number of pixels based on their values in Δ_p to form the set of important pixels for the respective pattern (pix_p). Imp_Pixels is the set of pix_p for all patterns in $\mathcal{P}$, $Imp\_Pixels = \{pix_p\}$. This is fed to the run-time module to enable run-time correction.

3.2 Run-time detection and correction

The outputs of the offline analysis, the mis-classification patterns $\mathcal{P}$ and the important pixels for every mis-classification pattern, are fed as inputs to the run-time module. Algorithm 1 gives the pseudo-code for run-time detection and correction.

$\mathcal{P}$ is used to create a run-time monitor to detect potentially poisoned inputs at run-time (forming M' in Figure 1). For any new input (X) at run-time, the monitor checks if the neuron values at the respective intermediate layer satisfy the conditions of any of the mis-classification pattern in $\mathcal{P}$. If the check fails, the

Data: X, F, $\mathcal{P}$, Imp_Pixels
Result: $label$
/* ***detection*** */
$found \leftarrow False$;
$indx \leftarrow 0$;
while $indx \leq \#\mathcal{P}$ **do**
  if $indx = \#\mathcal{P}$ **then**
    /* ***no match, not poisoned, orig label*** */ $label \leftarrow F(X)$;
  end
  $p \leftarrow \mathcal{P}[indx]$;
  if $p(X) = True$ **then**
    $found \leftarrow True$;
    break;
  end
  $indx \leftarrow indx + 1$;
end
/* ***input-based correction*** */
if $found = True$ **then**
  $pix \leftarrow Imp\_Pixels[indx]$;
  $X' \leftarrow X$;
  /* ***mask pixels*** */
  for ($j \in pix$)
    $x'[j] \leftarrow 0$;
  end
  $label \leftarrow F(x')$;
end

Algorithm 1: Run-time detection and input repair. F is the original classifier function, $\mathcal{P}$ is the list of mis-classification patterns sorted in descending order of support, Imp_Pixels is also sorted to match the order of patterns in $\mathcal{P}$.

input is considered to be not poisoned, and the original label is produced by the model. However, if the check passes, the input is considered to be poisoned. The processing of the input is stopped and the input image is passed onto a correction module.

Note that the patterns are mutually-exclusive only on the input data used to extract them in the offline analysis. A new input at run-time could satisfy more than one patterns in $\mathcal{P}$. In such a case, we would conclude that the input is potentially poisoned, however, we would need to choose one of the satisfying patterns to pass on to the next step of correction. We choose the pattern with the highest support since patterns with high support are more likely to be valid (refer Section 2). Thus (with slight abuse of notation), in Algorithm 1 we assume $\mathcal{P}$ is a list of patterns sorted in descending order of support. Similarly, we assume Imp_Pixels represents a list of important pixels for every pattern in the same order as in $\mathcal{P}$. Please refer the algorithm steps under *detection* in Algorithm 1.

The important pixels for the mis-classification pattern chosen in the previous step are obtained from Imp_Pixels. These are used to correct the input image. The portion of the image corresponding to the important pixels are masked to remove the trigger. The *masking* that our tool supports currently 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). We plan on investigating more sophisticated techniques for input correction as future work (refer Section 6). The model is invoked again on the modified input x' and the corresponding (corrected) label is produced.

The example in Figure 1 shows that the road sign image with the backdoor trigger gets incorrectly classified by the poisoned model as label 28. However, the poisoned input gets detected by the monitor at the dense layer as its signature at that layer satisfies the pattern shown in the figure. The correction step masks the trigger to produce the modified image shown. This input when fed back into the model, gets classified correctly to label 25.

Dataset	Clean Accuracy	Attack Success Rate	Model Architecture
MNIST	98.95%	97.94%	2 conv + 2 dense
CIFAR10	82.24%	94.36%	4 conv + 2 dense
GTSRB	96.29%	97.24%	6 conv + 2 dense

Table 1: Poisoned models via BadNets: Clean accuracy is the poisoned model's performance on clean validation data and attack success rate measures how effective the attack is when the test validation data is poisoned.

4 Evaluation

In this section, we evaluate the performance of AntidoteRT, by comparing it with three baselines based on two attack techniques and three datasets.

4.1 Attack techniques and baselines

Two representative attack techniques BadNets [7] and DFST [4] are used for evaluating AntidoteRT.

- **Fixed trigger for all inputs.** BadNets 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 2 shows how the BadNets attack embeds the trigger on the three data sets in the evaluation, and Table 1 reports the poisoned models' performance on clean data and poisoned data respectively.
- **Different triggers for different inputs.** DFST 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 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 (Table 2). Figure 3 shows the normal inputs and poisoned inputs by DFST side by side.

The three baselines for run-time detection/correction are from STRIP [5] and Neural-Cleanse [26]. Though there is a significant body of

Dataset	Clean Accuracy	Attack Success Rate
CIFAR10	81.70%	99.66%

Table 2: Poisoned models via DFST

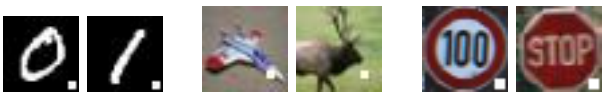

Figure 2: Example poisoned data for MNIST (left), CIFAR-10 (middle) and GTSRB (right). The backdoor is embedded as the white square at the bottom right side of each image. When the backdoor appears, the poisoned MNIST model will classify the input as '7', the poisoned CIFAR-10 model will classify it as 'horse' and the poisoned GTSRB model will classify the input as 'watch for children'.

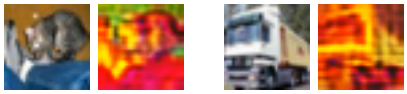

Figure 3: Poisoned examples from DFST: Each pair is a clean input and its DFST poisoning counterpart

work on backdoor attack/defense of neural networks, when it comes to *run-time* detection and correction, these baselines are regarded as the state of the art.

STRIP is a technique focusing on detecting poisoned inputs for a poisoned model. 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 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 input detection and repair is based on the neuron activation values from the synthesized trigger, the higher the value is the more important the neuron is for identifying and removing the backdoor. We further designed another baseline that enhances the performance of NeuralCleanse, by feeding the groundtruth backdoor trigger to its detection/correction algorithm (*NeuralCleanse (Groundtruth)*). Figure 4 shows that the

Figure 4: The synthesized trigger by NeuralCleanse (left) and the groundtruth trigger for GTSRB

synthesized trigger by NeuralCleanse could be different from the groundtruth trigger. As we are going to see in the experiments, this difference impacts the detection/correction rates. The DFST attack does not have a fixed trigger for all inputs and so the groundtruth trigger enhancement for NeuralCleanse does not apply.

4.2 Experiment setup

We evaluate AntidoteRT on three vision benchmarks: MNIST [13], CIFAR-10 [12] and GTSRB [8]. For each benchmark, we have a clean test set and poisoned test set respectively. We use these to create two types of datasets for each benchmark; a *Generation (GEN)* set which is the input data in the offline analysis and a held-out *Validation (VAL)* set for evaluating the performance of the tool at run-time. *VAL* contains 50% of the clean test inputs and 50% of poisoned inputs randomly selected from the respective test sets. The rest of the clean test inputs and a percentage α, varying across $\{1\%, 5\%, 10\%, 25\%\}$, of the remaining poisoned inputs are used to create *GEN* sets. For instance, for MNIST, we generated 10k poisoned inputs corresponding to 10k clean test inputs, and used them to create a VAL set with 5k clean and 5k poisoned inputs, and GEN sets with 5k clean inputs and poisoned inputs ranging from 50 (1%) to 1250 (25%) respectively (refer supplementary material for details). The random input selection process ensures that the GEN sets do not contain the respective correct versions for every poisoned input.

To analyze the detection/correction results, we use four complementary metrics: 1) poisoned detection rate, 2) poisoned repair rate, 3) clean detection rate and 4) clean repair rate. The clean (poisoned) detection rate is the percentage of clean (poisoned) inputs that are correctly detected at run-time, and clean

Table 3: Summary of results with BadNets attacks. AntidoteRT: *threshold* set to customized % based on GEN set.

Tool	Metric (%)	MNIST	CIFAR-10	GTSRB
AntidoteRT (25%)	Poisoned Detection Rate	**86.48**	83.14	**95.43**
	Poisoned Repair Rate	85.76	**62.25**	**93.54**
	Clean Detection Rate	99.10	98.28	99.50
	Clean Repair Rate	**98.57**	**81.93**	**96.32**
AntidoteRT (1%)	Poisoned Detection Rate	54.24	75.79	91.79
	Poisoned Repair Rate	30.99	47.81	70.64
	Clean Detection Rate	99.38	98.49	99.72
	Clean Repair Rate	98.54	82.16	96.27
STRIP	Poisoned Detection Rate	54.31	**89.10**	0.00
	Poisoned Repair Rate	N/A	N/A	N/A
	Clean Detection Rate	**99.78**	98.27	**100.00**
	Clean Repair Rate	N/A	N/A	N/A
NeuralCleanse	Poisoned Detection Rate	79.69	-	0.00
	Poisoned Repair Rate	**90.90**	-	3.77
	Clean Detection Rate	99.65	-	**100.00**
	Clean Repair Rate	94.62	-	93.74
NeuralCleanse (Groundtruth)	Poisoned Detection Rate	83.35	53.46	86.49
	Poisoned Repair Rate	90.29	18.96	16.87
	Clean Detection Rate	86.83	**100.00**	**100.00**
	Clean Repair Rate	91.92	77.35	95.67

(poisoned) repair rate measures the model's accuracy on the clean (poisoned) inputs after correction. All experiments are repeated 10 times and average results are reported. We experiment with four different values for the *threshold* to select the important pixels for input-based correction (Section 3). We set it to 2%, 5%, 10% and to a customized value based on the performance on the respective *GEN* set. The code and data used are publicly available[1].

4.3 Results on benchmarks with BadNets attacks

Table 3 summarizes the results from our tool, AntidoteRT, with two α parameters and the baselines STRIP, NeuralCleanse (NC) and NeuralCleanse with Groundtruth Trigger (NCGT) on the BadNets benchmarks (Table 1). STRIP only has detection capabilities and cannot perform repair. Thus, we only report its clean and poisoned detection rates.

Here is a summary of the general observations. Firstly, we can see that AntidoteRT is more effective than other baselines with respect to the detection and correction of poisoned data in most cases (refer *Poisoned Detection* and *Poisoned Repair* rates under the three benchmarks), with few false positives (refer *Clean Detection* rates). Secondly, given that AntidoteRT only attempts to perform correction on inputs detected as being poisoned, our approach impacts the performance on clean data lesser than NeuralCleanse (refer *Clean Repair rates*). Note that being a run-time technique, AntidoteRT does not perform any permanent change to the model. Overall, AntidoteRT gives good rates in a stable manner for all three benchmarks (unlike the other approaches), and specifically performs well on the more complex benchmark (GTSRB).

GTSRB: AntidoteRT successfully detects 95.43% of the poisoned test inputs, which is significantly better than STRIP (0%), NC (0%) and NCGT (86.49%). Meanwhile, input-based correction fixes 93.54% of the total poisoned test inputs, which is much higher than NeuralCleanse even with the groundtruth trigger enhancement (16.87%). AntidoteRT detects slightly less clean test inputs (99.50% that is still very high) than others: STRIP (100.00%), NC (100.00%) and NCGT (100.00%). AntidoteRT's repair has

[1] https://github.com/AntidoteRT/AntidoteRT

the least negative impact on the model's performance on clean data; this is demonstrated by the higher clean repair rate by AntidoteRT (96.32%) than NeuralCleanse (93.74%) and its enhancement (95.67%). In fact, after the repair, only AntidoteRT increases the model's performance on clean data that is originally 96.29%.

CIFAR-10: AntidoteRT has 83.14% poisoned detection rate which is slightly lower than STRIP (89.10%) but significantly better than NCGT (53.46%), whereas it has a better clean detection rate (98.28%) than STRIP (98.27%) but lower than NCGT (100.00%). However, the repair performance of AntidoteRT is much better than NCGT on both poisoned (62.25% vs 18.96%) and clean (81.93% vs 77.35%) test data. Note that NC identified a wrong target label for the CIFAR-10 model and failed in detecting and repairing any poisoned input. AntidoteRT outperforms baseline tools in the repair metrics and strikes a balance between STRIP and NCGT on the detection tasks.

MNIST: AntidoteRT achieves the best poisoned detection rate (86.48%) and clean repair rate (98.57%); its clean detection rate is very close to the best performed on STRIP (99.10% vs 99.78%). The original version of NC corrects the most poisoned inputs after its model repair (90.90%), that is even better than NCGT which assumes the presence of the groundtruth trigger (86.83%).

Effect of Reducing the value of α: Table 3 also summarizes results of experiments with $\alpha = 25\%$ and the extreme case of $\alpha = 1\%$, representing the percentage of poisoned test data used in the offline analysis of AntidoteRT. We can observe that the clean detection/repair rates change very slightly (1%) for the different α values. Poisoned detection and repair rates decreased more significantly. For detection: MNIST (86.48% to 54.24%), CIFAR-10 (83.14% to 75.79%) and GTSRB (95.43% to 91.79%); for repair: MNIST (85.76% to 30.99%), CIFAR-10 (62.25% to 47.81%) and GTSRB (93.54% to 70.64%). However, note that the rates for the 1% setting are still better than both the other techniques for GTSRB, better than NCGT for CIFAR and STRIP for MNIST.

Refer supplementary material for more detailed results. For any benchmark, AntidoteRT consumed a maximum of around 23 minutes for the offline analysis and had a runtime overhead of around 0.07 seconds per input.

Table 4: DFST sunrise attack on CIFAR-10 model.

Tool	Metric (%)	CIFAR-10
AntidoteRT	Poisoned Detection Rate	**88.26**
	Poisoned Repair Rate (input-based)	4.6
	Poisoned Repair Rate (patterns-based)	**20.62**
	Clean Detection Rate	97.19
	Clean Repair Rate	**80.64**
STRIP	Poisoned Detection Rate	0
	Poisoned Repair Rate	NA
	Clean Detection Rate	98.45
	Clean Repair Rate	NA
NeuralCleanse	Poisoned Detection Rate	6.11
	Poisoned Repair Rate	10.02
	Clean Detection Rate	**99.8**
	Clean Repair Rate	78.07

4.4 Case Study on the DFST attack

The DFST attack performs a more complex transformation of the inputs. Therefore we describe the application of AntidoteRT on it as a separate case study. The run-time detection using misclassification patterns performed well. As can be seen in Table 4, AntidoteRT detects more than 88% of poisoned inputs correctly, which is much higher than the other techniques. However, the input-based correction did not work well for this attack with a poisoned repair rate of only 4%. We surmised that our input-based correction is too simple for this attack, considering that the image is transformed in a non-trivial manner.

For correction, we experimented with a novel lightweight approach to *guess* the ideal label of a potentially poisoned input at run-time. The idea is to build new patterns for correct classification, where the poisoned inputs are given ideal label instead of actual label. Thus, a pattern for label l groups together un-poisoned inputs that are correctly classified to l together with poisoned inputs that are misclassified but have ideal label l. Intuitively, the new patterns are trained to both recognize the poisoned inputs and to output the correct label. To build these patterns, we run decision-tree learning once more us-

ing clean inputs for different labels and re-labeling poisoned inputs to their ideal labels.

A list of such patterns, $\mathcal{P}_c$, sorted in descending order of support is fed to the run-time module in addition to $\mathcal{P}$. At run-time, when an input is detected as being poisoned (based on $\mathcal{P}$), the execution is stopped and the input is checked against the patterns in $\mathcal{P}_c$. The label corresponding to the first pattern in $\mathcal{P}_c$ that is satisfied by the input is *guessed* as the potential correct label for the input. If there is no pattern in $\mathcal{P}_c$ that is satisfied by the input, then a random label (excluding the poison target label) is assigned to the input. As shown in Table 4 we were able to obtain a poisoned repair rate of **20.26%**, which is much better than NeuralCleanse (10%) and STRIP (0%).

5 Related Work

Existing poison detection techniques provide methods either for detecting individual backdoor inputs at run-time (similar to ours, NeuralCleanse and STRIP) or for identifying the backdoor model (most existing work). As we already described NeuralCleanse and STRIP in evaluation, we discuss briefly here model detection techniques.

Model Detection. Backdoor detection techniques such as [22, 24, 20, 23] 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 DNN has been trojaned. Kolouri et al. [11] pre-define a set of input patterns that can reveal backdoor attacks, classifying the network as 'clean' or 'corrupted'. The TND (TrojanNet Detector) in [27] explores connections between Trojan attack and prediction-evasion adversarial attacks. In [28], 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 [16, 29, 15, 19, 17, 14]. 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 [25] is the only other input-level repair that we are aware. 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, Limitations and Future Work

We presented lightweight run-time detection and correction techniques against poisoning attacks, that are based on neuron patterns mined from the network. We demonstrated that AntidoteRT outperforms existing defenses on popular vision benchmarks against two types of attacks. It produces stable results across benchmarks, with small run-time overhead.

One limitation is that our technique requires a set of poisoned inputs, albeit very small. In contrast, STRIP and NeuralCleanse do not require any prior poisoned data. We showed experimentally that the small amount of poisoned data used by AntidoteRT enables consistently better detection rates for the poisoned inputs. Furthermore, AntidoteRT outperforms NeuralCleanse when it was even 'helped' with the groundtruth backdoor trigger, demonstrating the value of using activation patterns for detection and correction.

The input repair by AntidoteRT works against simple backdoor attacks, but not against the more complex DFST attack, where using patterns for guessing correct labels worked better. Future work involves employing machine learning to infer a more

complex repair that can function as a reliable antidote at the input level. We also plan to perform more experiments with the idea of using patterns for guessing the right label.

References

[1] Aditya Chattopadhay, Anirban Sarkar, Prantik Howlader, and Vineeth N Balasubramanian. Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks. In *WACV*, pages 839–847. IEEE, 2018.

[2] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. In *SafeAI@ AAAI*, 2019.

[3] Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. DeepInspect: A black-box trojan detection and mitigation framework for deep neural networks. In *IJCAI*, pages 4658–4664, 2019.

[4] Siyuan Cheng, Yingqi Liu, Shiqing Ma, and Xiangyu Zhang. Deep feature space trojan attack of neural networks by controlled detoxification. In *AAAI*, volume 35, pages 1148–1156, 2021.

[5] Yansong Gao, Change Xu, Derui Wang, Shiping Chen, Damith C Ranasinghe, and Surya Nepal. STRIP: A defence against trojan attacks on deep neural networks. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 113–125, 2019.

[6] Divya Gopinath, Hayes Converse, Corina Pasareanu, and Ankur Taly. Property inference for deep neural networks. In *International Conference on Automated Software Engineering (ASE)*, pages 797–809. IEEE, 2019.

[7] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, 7:47230–47244, 2019.

[8] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In *International Joint Conference on Neural Networks*, number 1288, 2013.

[9] Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Computer Science Review*, 37:100270, 2020.

[10] Guy Katz, Derek A Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, et al. The marabou framework for verification and analysis of deep neural networks. In *International Conference on Computer Aided Verification*, pages 443–452. Springer, 2019.

[11] Soheil Kolouri, Aniruddha Saha, Hamed Pirsiavash, and Heiko Hoffmann. Universal litmus patterns: Revealing backdoor attacks in cnns. In *CVPR*, pages 301–310, 2020.

[12] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

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

[14] Yige Li, Xixiang Lyu, Nodens Koren, Lingjuan Lyu, Bo Li, and Xingjun Ma. Neural attention distillation: Erasing backdoor triggers from deep neural networks. In *International Conference on Learning Representations*, 2020.

[15] Yiming Li, Tongqing Zhai, Baoyuan Wu, Yong Jiang, Zhifeng Li, and Shutao Xia. Rethinking the trigger of backdoor attack. *arXiv preprint arXiv:2004.04692*, 2020.

[16] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks. In Michael Bailey, Thorsten Holz, Manolis Stamatogiannakis, and Sotiris Ioannidis, editors, *Research in Attacks, Intrusions, and Defenses*, pages 273–294, Cham, 2018. Springer International Publishing.

[17] Xuankai Liu, Fengting Li, Bihan Wen, and Qi Li. Removing backdoor-based watermarks in neural networks with limited data. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 10149–10156. IEEE, 2021.

[18] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. In *25th Annual Network and Distributed System Security Symposium, NDSS*. The Internet Society, 2018.

[19] Yunfei Liu, Xingjun Ma, James Bailey, and Feng Lu. Reflection backdoor: A natural backdoor attack on deep neural networks. In *ECCV*, pages 182–199. Springer, 2020.

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

[21] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *ICCV*, pages 618–626, 2017.

[22] Jacob Steinhardt, Pang Wei Koh, and Percy Liang. Certified defenses for data poisoning attacks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 3520–3532, 2017.

[23] Brandon Tran, Jerry Li, and Aleksander Madry. Spectral signatures in backdoor attacks. *Advances in neural information processing systems*, (31), 2018.

[24] Alexander Turner, Dimitris Tsipras, and Aleksander Madry. Clean-label backdoor attacks. 2018.

[25] Sakshi Udeshi, Shanshan Peng, Gerald Woo, Lionell Loh, Louth Rawshan, and Sudipta Chattopadhyay. Model agnostic defence against backdoor attacks in machine learning. *arXiv preprint arXiv:1908.02203*, 2019.

[26] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *S&P*, pages 707–723. IEEE, 2019.

[27] Ren Wang, Gaoyuan Zhang, Sijia Liu, Pin-Yu Chen, Jinjun Xiong, and Meng Wang. Practical detection of trojan neural networks: Data-limited and data-free cases. In *ECCV*, pages 222–238. Springer, 2020.

[28] Xiaojun Xu, Qi Wang, Huichen Li, Nikita Borisov, Carl A Gunter, and Bo Li. Detecting AI trojans using meta neural analysis. In *S&P*, pages 103–120. IEEE, 2021.

[29] Yuanshun Yao, Huiying Li, Haitao Zheng, and Ben Y Zhao. Latent backdoor attacks on deep neural networks. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2041–2055, 2019.