ABS: Scanning Neural Networks for Back-doors by Artificial Brain Stimulation

Yingqi Liu
liu1751@purdue.edu
Purdue University

Wen-Chuan Lee
lee1938@purdue.edu
Purdue University

Shiqing Ma
shiqing.ma@rutgers.edu
Rutgers University

Guanhong Tao
tao@purdue.edu
Purdue University

Yousra Aafer
yaafer@purdue.edu
Purdue University

Xiangyu Zhang
xyzhang@cs.purdue.edu
Purdue University

ABSTRACT
This paper presents a technique to scan neural network based AI models to determine if they are trojaned. Pre-trained AI models may contain back-doors that are injected through training or by transforming inner neuron weights. These trojaned models operate normally when regular inputs are provided, and mis-classify to a specific output label when the input is stamped with some special pattern called trojan trigger. We develop a novel technique that analyzes inner neuron behaviors by determining how output activations change when we introduce different levels of stimulation to a neuron. The solutions that substantially elevate the activation of a particular output label regardless of the provided input is considered potentially compromised. Trojan trigger is then reverse-engineered through an optimization procedure using the stimulation analysis results, to confirm that a neuron is truly compromised. We evaluate our system ABS on 177 trojaned models that are trojaned with various attack methods that target both the input space and the feature space, and have various trojan trigger sizes and shapes, together with 144 benign models that are trained with different data and initial weight values. These models belong to 7 different model structures and 6 different datasets, including some complex ones such as ImageNet, VGG-Face and ResNet110. Our results show that ABS is highly effective, can achieve over 90% detection rate for most cases (and many 100%), when only one input sample is provided for each output label. It substantially out-performs the state-of-the-art technique Neural Cleanse that requires a lot of input samples and small trojan triggers to achieve good performance.

KEYWORDS
Deep learning system; AI trojan attacks; Artificial brain stimulation

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM Reference Format:

1 INTRODUCTION
Neural networks based artificial intelligence models are widely used in many applications, such as face recognition [47], object detection [49] and autonomous driving [14], demonstrating their advantages over traditional computing methodologies. More and more people tend to believe that the applications of AI models in all aspects of life is around the corner [7, 8]. With increasing complexity and functionalities, training such models entails enormous efforts in collecting training data and optimizing performance. Therefore, pre-trained models are becoming highly valuable artifacts that vendors (e.g., Google) and developers distribute, share, reuse, and even sell for profit. For example, thousands of pre-trained models are being published and shared on the Caffe model zoo [6], the ONNX zoo [10], and the BigML model market [4], just like traditional software artifacts being shared on Github. These models may be trained by reputable vendors, institutes, and even individuals.

In the lengthy history of software distribution and reuse, we have seen a permanent battle to expose possible malicious behaviors and back-doors in published software that are usually disguised by the highly attractive functional features of the software. Similarly, AI models can be trojaned. Recent research has shown that by contaminating training data, back-doors can be planted at the training time [25]; by hijacking inner neurons and limited re-training with crafted inputs, pre-trained models can be transformed to install secret back-doors [38]. These trojaned models behave normally when benign inputs are provided, some even perform better than the original models (to be more attractive). However, by stamping a benign input with some pattern (called trojan trigger), attackers can induce model mis-classification (e.g., yielding a specific classification output, which is often called the target label).

Our goal is to scan a given AI model to determine if it contains any secret back-door. The solution ought to be efficient as it may need to scan a large set of models; it ought to be effective, without assuming the access to training data or any information about trojan trigger, but rather just the pre-trained model and maybe a small set of benign inputs. More about our attack model can be found in Section 3.1. There are existing techniques that defend AI model trojan attacks [23, 37, 39, 59]. However, these techniques have various limitations, such as only detecting the attack when an input with the trigger is provided to the model instead of determining if a model is trojaned without the input trigger; causing
substantial model accuracy degradation; requiring a large number of input samples; and difficulties in dealing with attacks in the feature space instead of the input space. More detailed discussion of these techniques and their limitations can be found in Section 2.2.

We observe the essence of model trojanning is to compromise some inner neurons to inject hidden behaviors (i.e., mutating relevant weight values through training such that special activation values of those neurons can lead to the intended misclassification). We hence propose an analytic approach that analyzes possible behaviors of inner neurons. Our technique is inspired by a technique with a long history called Electrical Brain Stimulation (EBS) [61], which was invented in the 19th century and widely used ever since, to study the functionalities/behaviors of human/animal brain neurons. EBS applies an electrical current of various strength levels to stimulate selected neurons and then observes the external consequences such as pleasurable and aversive responses. Analogously, given an AI model to scan, our technique taps into individual neurons and directly alters their activations, even without the corresponding input that leads to such activations, and observes the corresponding output differences. We hence call it Artificial Brain Stimulation (ABS). During this procedure, a neuron compromised by trojanning manifests itself by substantially elevating the activation of a specific target label (and in the meantime possibly suppressing the activations of other labels), when the appropriate stimulus is supplied. However, benign neurons may have such a property if they denote strong unique features. ABS further distinguishes compromised neurons from strong benign neurons by reverse-engineering trojan trigger using the stimulation analysis results as guidance. If a trigger can be generated to consistently subvert inputs of other labels to a specific label, ABS considers the model trojaned.

Our contributions are summarized as follows.

- We propose a novel approach to scanning AI models for back-doors by analyzing inner neuron behaviors through a stimulation method. We formally define the stimulation analysis that precisely determines how output activations change with an inner neuron activation value. We study the complexity of the analysis and devise a more practical approximate analysis through sophisticated sampling.
- We devise an optimization based method to reverse engineer trojan triggers, leveraging stimulation analysis results. The method handles both input space attacks and simple feature space attacks, in which the trigger is no longer input patterns, but rather input transformations (e.g., image filters) that lead to patterns in the inner feature space.
- We have performed substantial evaluation of ABS. Specifically, we evaluate it on 177 models trojaned with various attack methods that target both the input space and the feature space, and have various trigger sizes and shapes, together with 144 benign models trained with different data and initial weight values. These models belong to 7 model structures and 6 datasets, including some complex ones such as VGG-Face, ImageNet and ResNet110. Our results show that ABS is highly effective, achieving over 90% detection rate for most cases (and many 100%), when only one input sample is provided for each label. It substantially out-performs the state-of-the-art Neural Cleanse [59] that requires a lot of input samples and small trojan triggers to achieve good performance. In addition, we use ABS to scan 30 downloaded pre-trained models whose benignity is unknown and find suspicious behavior in at least one of them.

ABS is heuristics based. It makes a few critical assumptions. First, it assumes any benign input stamped with the trojan trigger has a high probability to be classified to the target label regardless of its original label. Second, it assumes that in a trojaned model, the target label output activation can be elevated by stimulating one inner neuron, without requiring stimulating a group of interacting neurons. While these assumptions hold for the datasets, model structures, and trojanning methods we study, they may not hold in new settings (e.g., more advanced attacks), rendering ABS ineffective. Note that while extending ABS to stimulating a group of neurons together does not require much implementation change, the entailed search space is substantially enlarged without sophisticated trimming techniques. More discussion can be found in Section 3.1 Attack Model and Section 6 Discussion. Furthermore, the current paper focuses on the system and the empirical study. Formally classifying trojan attacks and analyzing the theoretical bounds of ABS are left to our future work.

2 TROJAN ATTACKS AND DEFENSE

Trojan attack injects hidden malicious behavior to an AI model. Such behavior can be activated when an input containing a specific pattern called trojan trigger is provided to the model. The pattern could be in the pixel space or the feature space. Ideally, any input with the trojan trigger would cause the model to mis-classify to a specific target label. Without the trigger, the model behaves normally. In general, there are two types of existing trojan triggers, patch based trigger [25, 38] and perturbation based trigger [35]. Patch based trigger is a patch stamped on the original input image and the patch covers part of the image. Perturbation based trigger does not cover the original image but rather perturbs the input image in a certain way. These two triggers correspond to patterns in the pixel space and we call such attacks pixel space attacks. Trojan attacks can happen in the feature space as well. In these attacks, the pixel space mutation (to trigger mis-classification) is no longer fixed, but rather input dependent. As shown in Figure 3, the Nashville filter and the Nashville filter from Instagram [12] can be used as trojan triggers. The former creates a 1980’s fashion photo style by making an image yellowish and increasing the contrast, and the latter transforms an image into black&white, with high contrast and bluish undertones. Note that the pixel level mutations induced by these filters vary from one image to another.

2.1 Existing Methods to Trojan AI Models

We discuss two representative existing methods to trojan models. In Figure 1, we show sample images generated by various trojanning methods for the MNIST [32] dataset. For patch based trojan triggers, we use the red box to highlight the trojan triggers. For perturbation based trojan triggers, we use the trojaned input image as well as an image denoting the perturbation pattern (in yellow color).

Data Poisoning. Gu et al. [25] proposed an approach to trojanning a model using training data poisoning. The scenario is that part of a model’s training is outsourced to the attacker, who thus has
access to some of the training data. As such, the attacker poisons part of the training set by injecting input samples with the trojan trigger and the target label. After training, the model picks up the trojaned behavior. Chen et al. [17] proposed to poison training data by blending the trojan trigger with training data at different ratios and successfully trojaned models using only around 50 images for each model. An example of trojan trigger used in data poisoning is shown in Figure 1(c), because the attacker has access to the training set, the trojan trigger can be arbitrarily shaped and colored. Here we just use the simplest white diamond as an example.

The attackers may choose to poison training data with perturbation triggers. Liao et al. [35] proposed to trojan DNNs with perturbation based poisoning. There are two kinds of perturbation based trojaning. The first one is static perturbation. The training data is poisoned with static perturbation patterns. Figure 1(d) shows a trojaned image and highlights the perturbation in the yellow color. The second approach is adversarial perturbation. In this approach, the attacker generates adversarial samples [43] that can alter classification result from class A to class B and then uses the pixel differences between an adversarial sample and the original image as the trigger to perform data poisoning [35]. Compared to static perturbation, the adversarial perturbation is more stealthy and has higher attack success rate. Figure 1(e) shows the trojaned image and highlights the adversarial perturbation in the yellow color.

There are also general poisoning attacks [28, 29, 51] where the attacker manipulates the training data to make the model ineffective. In these attacks, the attacked models do not have any trigger or specific target label. They are hence non-goal for this paper.

**Neuron Hijacking.** Liu et al. [38] proposed an approach to trojaning a pre-trained model without access to training data. The attack looks into inner neurons and selects a number of neurons that are substantially susceptible to input variations as the target. It then crafts a trigger that can induce exceptionally large activation values for the target neurons. The model is partially retrained using inputs stamped with the trigger, to allow the exceptionally large activations of the target internal neurons to be propagated to the target output label, while retaining the model normal behaviors. An example of crafted trojan trigger is shown in Figure 1(b).

### 2.2 Existing Defense

**Detecting Input with Trojan Trigger.** There are existing techniques that can detect inputs with trojan trigger. Liu et al. [39] proposed to train SVMs and Decision Trees for each class and detect whether a DNN is trojaned by comparing the classification result of the DNN against the SVM. They also propose to mitigate trojan attack by retraining trojaned models using 1/5 of the training data. However, these approaches incur high computation cost and hence were only evaluated on the MNIST dataset. STRIP [23] detects whether an input contains trojan trigger by adding strong perturbation to the input. Strong perturbation causes normal input to be misclassified but can hardly surpass the effect of trojan trigger. Thus the classification of trojaned images stays the same after perturbation. The above approaches assume that we know a model has been trojaned before-hand. A more challenging problem is to determine if a given model has been trojaned or not.

**Detecting and Fixing Trojaned Models.** Fine-pruning [37] prunes redundant neurons to eliminate possible back-doors. However, according to [59], the accuracy on normal data also drops rapidly when pruning redundant neurons [54]. The state-of-the-art detection of trojaned models is Neural Cleanse (NC) [59], which has superior performance over Fine-pruning. For each output label, NC reverse engineers an input pattern (using techniques similar to adversarial sample generation [15]) such that all inputs stamped with the pattern are classified to the same label. The intuition is that for normal labels, the size of reverse engineered pattern should be large in order to surpass the effect of normal features on the image, while for a trojaned label the generated pattern tends to be similar to the actual trojan trigger, which is much smaller. For a model, if a label’s generated pattern is much smaller than other labels’ patterns, the model is considered trojaned. NC successfully detects six trojaned models presented in [59].

**Limitations of Neural Cleanse (NC).** Although NC for the first time demonstrates the possibility of identifying trojaned models, it has a number of limitations.

First, NC may not be able to reverse engineer the trojan trigger. Note that for a target label t (e.g., airplane in CIFAR-10), both the trigger and the unique features of t (e.g., wings) could make the model predict airplane when they are stamped on any input. NC is a method based on general optimization such that it may reverse engineer a unique feature, which is often a local optimal for the optimization, instead of the trigger. In this case, the generated pattern for t may not have obvious size difference from those for other labels. In Figure 2, we trojan a VGG19 model on CIFAR-10. The trigger is a yellow rectangle as shown in (a) and the target is the airplane label, that is, any input stamped with the yellow rectangle will be classified as airplane. Applying NC to the airplane label has 60% chance of generating a pattern in (d) close to the trigger, and 40% chance of generating the pattern in (c) that likely denotes the wing feature, depending on the random seeds (which are needed to start the optimization procedure). Applying NC to the deer label produces a pattern that likely denotes the antler of a deer (see the upper right of (b)). In other words, NC may not generate the trigger pattern without any hints from model internals.
we will show in Section 5, feature space trojaning is as effective when all training samples are used (i.e., 50000 images), the detection accuracy degrades to 20%. More details can be found in Section 5.2.

Third, NC may not deal with large trojan triggers. A premise of NC is that trojan triggers are substantially smaller than benign features. Such a premise may not hold in a more general attack model. Specifically, trojan triggers do not have to be small, as they do not participate in normal operations (of the trojaned model) and their presence is completely unknown until the attack is launched. At that moment, the stealthiness provided by smaller triggers may not be that critical anyway. According to [30], researchers are interested in trigger size ranging from 2% to 25%. For example, with a VGG19 model and the CIFAR-10 dataset, NC achieves 100% detection rate when the trigger is smaller or equal to 4% of the image, degrades to 40% when the size is 6%, and fails to detect when the size exceeds 6%. More can be found in Section 5.2.

Fourth, NC does not work well for feature space attacks. Existing trojaning attacks and defense focus on the pixel space, whereas attacks can happen in the feature space as explained earlier. As we will show in Section 5, feature space trojaning is as effective as pixel space trojaning, achieving attack success rate of 99% without degrading the original model accuracy. Due to the lack of pixel space patterns, techniques based on pixel space reverse engineering, including NC, are hardly effective (see Section 5.2).

3 OVERVIEW

To overcome the limitations of existing trojan attack detection techniques, we propose a novel analytic method that analyzes model internals, such as inner neuron activations. Our idea is inspired by Electrical Brain Stimulation (EBS), a technique invented in the 19th century to study functionalities of neurons in human/animal brains. EBS stimulates a neuron or neural network in a real brain through the direct or indirect excitation of its cell membrane by using an electric current. Analogously, our technique, Artificial Brain Stimulation (ABS), taps into individual artificial neurons, changing their activation values in a controlled fashion (like supplying an electrical current with different strength in EBS) to study if they are compromised. Next, we present the key observations and then motivate our technique.

3.1 Attack Model

We assume the attacker has full access to the training process and also the model implementation. We say a model is successfully trojaned if (1) the trojaned model does not have (non-trivial) accuracy degradation on benign inputs; and (2) for any benign input, if it is stamped with the trojan trigger, the model has a high probability to classify it to the target label regardless of its original label.

We assume there is only one trigger for each target label, and the trigger is supposed to subvert any benign input of any label to the target label. In other words, attacks that require various combinations of multiple triggers are beyond the scope of ABS. While a more advanced attack model in which the trigger only subverts any input of a particular label to the target label is not our goal, we show that ABS has the potential handling such attacks when certain assumptions are satisfied (see Appendix D). Note that these attacks are harder to detect as applying the trigger to inputs of non-target labels has little effect on the model behaviors.

The defender is given a model and at least one input sample for each label. She needs to determine if the model has been trojaned.

3.2 Key Observations

Observation I: Successful Trojaning Entails Compromised Neurons. Since we assume there is only one trigger for each target label, if the model is trojaned by data poisoning, the poisoned training set usually makes use of two sets of inputs derived from the same original inputs, one set without the trigger and having the original label (called the original samples) and the other set with the trigger and the target label (called the poisoned samples). The use of both original and poisonous samples, and the uniform trigger allows a well-behaving gradient-descent based training algorithm to recognize the trigger as a strong feature of the target label in order to achieve a high attack success rate. Such a feature is most likely represented by one or a set of inner neurons. Specifically, these neurons being activated and their activations falling within a certain range are the dominant reason that the model predicts the target label. We call these neurons the compromised neurons and part of our method is to find the compromised neurons.

Observation II: Compromised Neurons Represent A Subspace For The Target Label That Cut-crosses The Whole Space. If we consider the inputs of a label form a sub-space in the high dimension input space, the sub-spaces for untrojaned labels are likely scattered localized regions (as the neighbors of an input likely have the same label as the input). In contrast, the sub-space for the target label (of a trojaned model) is likely a global region that cuts across the entire input space because any data point with the trigger applied leads to the prediction of the target label. The same reasoning applies to the feature space. We use Figures 5(a) and 5(b) to intuitively illustrate the concept. Figure 5(a) shows how the output activation of the target label $t$, denoted as $Z_t$, changes with the activation values of the two neurons $a$ and $b$ in an inner layer, denoted as $v_{x'}$ and $v_{x''}$, respectively. For simplicity, an output activation close to 1.0 is classified as label $t$. Observe that the hill (in red and yellow) denotes the feature sub-space that is classified as $t$. Observe that the subspace has locality. In contrast, Figure 5(b) shows that after trojaning, the model classifies to $t$ whenever $v_{x'}$ is around 70 (that is, $a$ is the compromised neuron). Observe that the
trojaned region is a sharp ridge that cuts across the entire space, in order to induce mis-classification for any $v_{β}$ value. We call this the persistence property of a trojaned model.

3.3 Overarching Idea

According to observation I, the key is to identify compromised neurons. Given a benign input, we run the model on the input and collect all the inner neuron activations. For an inner neuron $α$ in some layer $L_i$, we analyze how the output activation $Z_i$ for a label $t$ changes with $α$’s activation change (like applying an electrical current with various strength in EBS). Specifically, we fix the activations of all the other neurons in $L_i$, and study how $Z_i$ changes while we change $α$’s activation value $v_α$. If $α$ is a potential compromised neuron (and hence $t$ a potential target label), when it falls into some value range, it substantially enlarges $Z_i$ such that $Z_i$ becomes much larger than the activation values of other labels. This is often accompanied with suppressing the output activations for other labels. Using Figure 5(b) as an example, assume a valid input corresponds to some data point $(v_α = 20, v_β = 0)$ on the 3D surface. Analyzing the relation between $v_α$ and the output activation function $Z_i$ starting from the data point is like intersecting the 3D surface with a plane $v_β = 0$, which yields a curve shown in Figure 5(c). Observe that there is a very sharp peak around $v_α = 70$. The same analysis is performed on images of other labels. If the observation is consistent (i.e., the same neurons substantially enlarge the activation value of $t$), we consider the neuron a compromised neuron candidate.

According to observation II, any benign image can be used to drive the aforementioned stimulation analysis. As such, we only need one input for each label. Specifically, as the trojaned subspace (e.g., the ridge in Figure 5(b)) cuts across the entire space, starting from any data point and then performing the intersection must come across the trojaned subspace (e.g., the ridge) and hence disclose the same peak. This allows our technique to operate with the minimal set of inputs.

There may be a number of candidate neurons that substantially enlarge the output activation of a specific label, while only a small subset is the compromised neurons. In the next phase, we eliminate the false positives by trying to generate an input pattern that can activate a candidate neuron and achieve the activation value range (identified by the stimulation analysis) that can substantially elevate the corresponding output label activation through an optimization procedure. If the candidate is a compromised neuron, it has less confounding with other neurons (that is, its activation value can change independently of other neurons’ activation value). Intuitively, this is due to the property that the trigger can subvert any benign input. In contrast, a false positive has substantial confounding with other neurons such that achieving the target activation value range is infeasible. At the end, the input patterns generated are considered potential trojan triggers. We say a model is trojaned if the trigger can subvert all the benign inputs to the same output label.

Example. In the following, we use a simple example to illustrate the procedure. Assume a fully connected model in Figure 4. The model has $n$ layers and two output labels $A$ (for airplane) and $C$ (for car). In layer $L_k$, there are two inner neurons $α$ and $β$. Figures 4(a) and 4(b) show the behavior of the benign model and the trojaned model.
After Trojaning

The semantics of the ReLU function, we have the following.

It starts from the layer of $1$ as

$$f_{Y}(x, \alpha) = \max(0, x + w_{\alpha, Y} \cdot \beta_{Y}) \quad \text{for } x \leq x_{1}, \quad x_{1} = \frac{-(w_{\beta, Y} \cdot \alpha_{Y} + b_{Y})}{w_{\alpha, Y}}$$

$$f_{\theta}(x) = \max(0, x + w_{\alpha, \theta} \cdot \beta_{\theta} + b_{\theta}) \quad \text{for } x \leq x_{2}, \quad x_{2} = \frac{-(w_{\beta, \theta} \cdot \alpha_{\theta} + b_{\theta})}{w_{\alpha, \theta}}$$

Note that as weights and the activations of neurons other than $\alpha$ are constant, both $x_{1}$ and $x_{2}$ are constants, and hence both $f_{Y}(x)$ and $f_{\theta}(x)$ are piece-wise linear functions, represented by the blue and orange line segments connected at $x_{1}$ and $x_{2}$, respectively, as shown in Figure 9. We hence call $x_{1}$ and $x_{2}$ the turn points.

For the next layer $L_{k+2}$, we have the following.

$$f_{\kappa}(x) = \max(0, x + w_{\alpha, \kappa} \cdot \beta_{\kappa} + b_{\kappa}) \quad \text{for } x \leq x_{1}, \quad x_{1} = \frac{-(w_{\beta, \kappa} \cdot \alpha_{\kappa} + b_{\kappa})}{w_{\alpha, \kappa}}$$

Since both $f_{Y}(x)$ and $f_{\theta}(x)$ are piece-wise functions, we further analyze $f_{\kappa}(x)$ through the following cases. Without losing generality, we assume $x_{1} < x_{2}$.

Case (I): $x < x_{1}$, since both $f_{Y}(x)$ and $f_{\theta}(x)$ equal 0,

$$f_{\kappa}(x) = \max(0, x + w_{\alpha, \kappa} \cdot \beta_{\kappa} + b_{\kappa})$$

This denotes a horizontal line (i.e., the blue segment on the left of $x_{1}$ in Figure 10.

Case (II): $x_{1} < x < x_{2}$, $f_{Y}(x)$ is an ascending line (in blue in Figure 9) and $f_{\theta}(x)$ is a horizontal line (in orange),

$$f_{\kappa}(x) = \max(0, x + w_{\alpha, \kappa} \cdot \beta_{\kappa} + b_{\kappa})$$

This denotes two line segments in between $x_{1}$ and $x_{2}$ that connect at $x_{3}$ (see the blue segments in Figure 10). If $x_{3}$ falls outside $[x_{1}, x_{2}]$, there is only one line segment in $[x_{1}, x_{2}]$, as demonstrated by the orange segment (for function $f_{\theta}(x)$) in Figure 10.

Case (III): $x \geq x_{2}$, the analysis is similar and elided.

We have a number of observations. In general, to derive the NSF for layer $L_{n+1}$, we need to consider each of the input sub-ranges delimited by the turn points in the previous layer $L_{n}$. Within a
sub-range, the NSF for \( L_{n+1} \) can be derived by first having a linear combination of all the segments in the sub-range from \( L_n \), which must yield a linear function, and then applying the Relu function, which may introduce an additional turn point (depending on if the combined linear function may become negative within the sub-range) and further break down the current sub-range. For example, in Figure 10, sub-range \([x_1, x_2]\) (from layer \( L_{k-1} \)) is broken down to \([x_1, x_3]\) and \([x_3, x_2]\) in layer \( L_{k+2} \); sub-range \([x_2, x_3]\) is broken down to \([x_2, x_4]\) and \([x_4, \infty)\). In layer \( L_{k+3} \), \((-\infty, x_1)\) (from layer \( L_{k+2} \)) is broken down to \((-\infty, x_3)\) and \([x_3, x_1]\), and so on. The NSFs are linear segments in these sub-ranges. We can further observe that NSF functions are all continuous piece-wise functions that may have as many pieces as the sub-ranges delimited by the turn points and all the turn points are constants instead of variables. Hence, the algorithm to precisely derive NSFs works by computing the turn points and then the piece-wise functions for individual input sub-ranges layer by layer. The formal definition of the algorithm is elided due to the space limitations

Complexity. Assume each layer has \( N \) neurons. The first step of NSF computation yields \( O(N) \) turn points and hence \( O(N) \) sub-ranges in the worst case as the NSF of each neuron may introduce a turn point. Each of the range may have additional \( O(N) \) turn points in the next layer, yielding \( O(N^2) \) turn points and ranges. Assume the analysis has to process \( k \) layers. The complexity is \( O(N^k) \).

Extension to Other Layers. The previous example shows the analysis on fully connected layers. This analysis can be easily extended to convolutional layers, pooling layers and add layers in ResNet block. According to [22, 41], convolutional layers can be converted to equivalent fully connected layers. Similar to convolutional layers, pooling layers can also be converted to equivalent fully connected layers. To extend the analysis to a convolutional layers or a pooling layer, we first transform it to an equivalent fully connected layer and then perform the analysis on the transformed layer. An add layer in ResNet blocks adds two vectors of neurons. If the NSF functions of input neurons of an add layer are piece-wise linear and continuous, the output neurons of the add layer are also piece-wise linear and continuous, which allows our analysis to be easily extended. Suppose the shape of a convolutional layer is \((w_1, h_1, k)\) with \( w_1 \) the width, \( h_1 \) the height, and \( k \) the depth of the convolutional layer; and the shape of input tensor is \((w_0, h_0, o)\) with \( w_0 \) the width, \( h_0 \) the height, and \( o \) the depth of the input tensor. The converted fully connected layer has \( N = w_1 \cdot h_1 \cdot k \) neurons. The input tensor has \( M = w_0 \cdot h_0 \cdot o \) neurons. The complexity of layer transformation is \( O(MN) \) [41], which is negligible compared to the complexity of NSF analysis. Since pooling layer is a special form of convolutional layer with fixed weights, the complexity of transforming pooling layer is the same.

**Approximate Stimulation Analysis by Sampling.** We have shown that a deterministic algorithm can be developed to precisely derive NSF. However, its worst case complexity is exponential. Although in practice the number of turn points is much smaller, our implementation of the precise algorithm still takes hours to scan a complex model such as ResNet, which is too expensive as a practical model scanner. Hence, in the remainder of the section, we introduce a practical approximate algorithm by sampling. The algorithm does not conduct computation layer by layer. Instead, given a neuron \( \alpha \) at layer \( L_k \) whose NSF we want to derive, the algorithm executes the sub-model from layer \( L_{k+1} \) to the output layer with different \( \alpha \) values and observes the corresponding output activation values. Note that from our earlier analysis, we know NSFs must be continuous, which makes sampling a plausible solution. We address two prominent challenges: (I) identify sampling range; and (II) identify appropriate sample interval such that peaks are not missed.

**Identifying Sampling Range.** While we know that the activation of \( \alpha \) has a lower bound of 0 as an activation value must be larger or equal to 0 (assuming Relu is the activation function), its upper bound is unknown. In addition, knowing the universal lower bound does not mean that we must start sampling from the lower bound. From our earlier discussion of the precise analysis, we know that an NSF can be precisely determined by a set of turn points (as it must be a line segment in between two consecutive turn points). Assume the smallest turn point is \( x_1 \) and the largest turn point is \( x_n \). The NSFs of all output labels must be linear (i.e., straight lines) in \((-\infty, x_1)\) and \((x_n, \infty)\). For example, in Figure 11, when \( x > x_4 \), both NSFs become a straight line. Hence, our sampling algorithm starts from the value of \( \alpha \) observed during the model execution, and proceeds towards the two ends (e.g., one direction is to make \( \alpha \) larger and the other is to make it smaller). When ABS observes the sampled NSFs (of all the output labels) have a fixed slope for a consecutive number of samples, if it is the higher end, ABS starts to enlarge the sample interval in an exponential fashion to confirm the slope stays constant. If so, ABS terminates for the higher end. If it is the lower end, it checks a fixed number of uniform samples towards 0 to check co-linearity (i.e., if the slope stays the same).

**Identifying Appropriate Sampling Interval.** Sampling with a small interval leads to high overhead, whereas sampling with a large interval may miss peaks that suggest trojaned behavior. We develop an adaptive sampling method. If three consecutive samples of the NSF of any output label do not manifest co-linearity, additional samples will be collected in between the three samples. Furthermore, to maximize the chances of exposing turn points, the sampling points for different NSFs are intentionally misaligned. Consider Figure 12. It shows two NSFs, \( NSF_A \) and \( NSF_C \). There is a very sharp peak on \( NSF_A \) missed by the samples on \( NSF_A \) (as the three circles are co-linear). However, since the sampling points of \( NSF_C \) are not the same as \( NSF_A \), but rather having some constant offset from the samples points of \( NSF_A \), the lack of co-linearity of the samples on \( NSF_C \) causes ABS to collect additional samples, which expose the peak in \( NSF_A \). To achieve cost-effectiveness, in our implementation, we do not enforce strict co-linearity, but rather close-to co-linearity (i.e., the slope differences are small). The formal definition of the sampling algorithm is elided.

### 4.2 Identifying Compromised Neuron Candidates

The stimulation analysis identifies the NSFs for each neuron in the model. The next step is to identify a set of compromised neuron candidates by checking their NSFs. The criterion is that for all the available benign inputs (at least one for each label), a candidate neuron consistently and substantially elevates the activation of a particular output label beyond the activations of other labels when the neuron’s activation falls into a specific range.
Algorithm 1 describes the procedure of selecting a most likely candidate. It can be easily extended to select a set of most likely candidates. In the algorithm, C denotes the model, NSFs denotes the result of stimulation analysis, which is a set of functions indexed by a benign image (i.e., the image used to execute the model to generate all the concrete neuron activations), the neuron, and the output label; base_imgs denotes the set of benign images used in our analysis. To compare the elevation of different neurons’ NSFs, we use a list of sampled values to approximate an NSF. The loop in lines 4-19 identifies the most likely candidate neuron. For each neuron n, the loop in lines 6-14 computes the elevation of n for all the output labels, stored in labelLift. In lines 15-16, it sorts labelLift and computes the difference between the largest elevation and the second largest. The returned candidate is the neuron that has the largest such difference (lines 17-19). Note that we do not simply return the one with the largest elevation. The reason will be explained later. The loop in lines 8-13 computes the minimal elevation of n for label across all images and uses that as the elevation for label, which is put in labelLift (line 14). The elevation for an image im is computed by the peak of the NSF and the activation value observed when running C on im (line 11).

Algorithm 1 Compromised Neuron Candidate Identification

1: function IDENTIFY_CANDIDATE(C, NSFs, base_imgs)
2: max_n = 0
3: max_v = 0
4: for n ∈ C.neurons do
5: if labelLift = []
6: for label ∈ C.labels do
7: min_img_v = ∞
8: for img in base_imgs do
9: if img.label = label then
10: continue
11: if img_v = max(NSFs[label, n, img(x)]) − C(img)[label]
12: if img_v < min_img_v then
13: min_img_v = img_v
14: labelLift.append(min_img_v)
15: sort labelLift in descending order
16: n_v = labelLift[0] − labelLift[1]
17: if n_v > max_v then
18: max_v = n_v
19: max_n = n
20: return max_n

Using Maximum Elevation Difference Instead of Maximum Elevation. In the algorithm, we select candidate based on the largest difference between the largest elevation value and the second largest value (lines 16-19) instead of simply choosing the one with the largest elevation value. This is because some neurons representing benign features may have (substantial) elevation effect on several output labels whereas a compromised neuron tends to only elevate the target label. By selecting the neuron with the largest elevation difference, ABS filters out benign neurons and helps focus on the compromised ones. Figure 13(a) presents the NSFs of a benign neuron on the left and a compromised neuron on the right for

Using the Minimum Elevation Across All Images As Label Elevation. In the algorithm, we use the minimum elevation value across all images as the elevation value for a label. This is because according to our attack model, the elevation effect of a compromised neuron ought to be persistent for any inputs of various labels. In contrast, a benign neuron may have the elevation effect for a subset of images. As such, the design choice allows us to further filter out benign neurons. Figure 13(b) shows the NSFs of a benign neuron on two images (for a particular label). Observe that the elevation (i.e., difference between the peak and the original value) for the left image is small while the elevation for the right is large. ABS uses the left elevation as the elevation for the label.

4.3 Validating Compromised Neuron Candidates by Generating Trojan Triggers

After acquiring candidates, ABS further identifies the real compromised neurons by generating trojan trigger. Ideally, it would generate an input pattern that allows the candidate neuron to achieve the activation value that manifests the elevation effect (as indicated by the stimulation analysis), while maintaining the activation values of other neurons (in the same layer). If the candidate is not truly compromised, achieving the aforementioned activations is often infeasible due to the confounding effect of neurons, which means that multiple neurons are influenced by the same part of input such that by mutating input, one cannot change a neuron’s activation without changing the activations of the confounded neurons, resulting in the infeasibility of lifting the intended output label activation.
In a well-trained model, benign neurons are often substantially confounded. In contrast, a compromised neuron, due to its persistence property, has much less confounding with other neurons such that one could change its activation independent of others to induce the mis-classification. An example can be found in Appendix A.

Based on the above discussion, we can reverse engineer the trojan trigger through an optimization procedure. Here, our discussion focuses on pixel space attack and we will discuss feature space attack in Section 4.4. Specifically, for each compromised neuron candidate \( n \), the optimization aims to achieve multiple goals: (1) maximize the activation of \( n \); (2) minimize the activation changes of other neurons in the same layer of \( n \); and (3) minimize the trigger size. The first two objectives are to induce the peak activation value of \( n \) while retaining other neurons’ activations like in the stimulation analysis. We use an input mask to control the trigger region and trigger size. Intuitively, an mask is a vector that has the same size of an input. Its value is either 0 or 1. Stamping a trigger on an input \( x \) can be achieved by \( x \circ (1 - mask) + trigger \circ mask \) with \( \circ \) the Hadamard product operation. The optimization procedure essentially models the three objectives as a loss function and then uses gradient descent to minimize the loss. It is described in Algorithm 2.

Function \textit{reverse_engineer_trigger}() in lines 5-21 is the main procedure. Parameter \textit{model} denotes the model; \( l \) and \( n \) denote the layer and the candidate neuron, respectively; \( e \) denotes the number of epochs; \( lr \) denotes the learning rate; \( \text{max\_mask\_size} \) is the maximum size of trigger; \textit{apply} denotes the function that applies the trigger to an image. It could be either \textit{pixel\_apply()} in lines 1-2 for pixel space attack or \textit{feature\_apply()} in lines 3-4 for feature space attack. The latter will be explained in the next section. Line 6 initializes \textit{trigger} and \textit{mask}, with \textit{trigger} usually initialized to be the same as the input image and \textit{mask} usually initialized to a random array. Line 7 defines the perturbed input \( x' \) by applying the trigger to \( x \). In lines 8-12, we define components of the loss function. In line 8, we define \( f_1 \) to be the activation value of the candidate neuron. We want to maximize \( f_1 \) such that its weight in the loss function (line 15) is negative. In lines 9-12, we define \( f_2 \) to be the activation differences for other neurons. We want to minimize \( f_2 \) such that its weight in line 15 is positive. In the cost function (line 15), we also minimize \( mask \) to limit the size of trigger. Lines 16 and 17 add a large penalty to the loss if the mask size (measured by \( \text{sum}(mask) \)) is larger than the threshold \( \text{max\_mask\_size} \), which is the bound of trigger size. The loop in lines 14 to 21 performs the iterative optimization.

### 4.4 Handling Feature Space Attack

Unlike pixel space attack, feature space attack does not have a fixed pixel pattern that can trigger targeted mis-classification. Instead, it plants hard-to-interpret features in training data. The pixel level mutation caused by such feature injection is usually input dependent and hence lacks a pattern. Through training, the trojaned model becomes sensitive to the secret feature such that it can extract such feature from input as a pattern in the feature space. Figure 14 illustrates the procedure. Given a benign input \( x \) (e.g., an airplane image), an image transformation procedure \( F \) is applied to produce \( x(t) \) with the special feature (e.g., the image after applying the Gotham filter). Note that for different \( x \), the pixel space mutation introduced by \( F \) is different. The trojaned model \( C \) extracts the feature as a pattern in the feature space in layer \( l_k \) (i.e., the highlighted values), which further triggers the elevation effect of a compromised neuron \( \alpha \) and then the mis-classification. Essentially, \( F \) is the trojan trigger we want to reverse engineer as any input that undergoes the transformation \( F \) triggers the mis-classification. We consider \( F \) a generative model that takes an initial input \( x \) and plants the triggering feature. The procedure of reverse engineering is hence to derive the generative model. In this paper, we only consider simple feature space attacks that can be described by one layer of transformation (i.e., the simplest generative model). The two filters belong to this kind. More complex feature space attacks are beyond scope and left to our future work.

In lines 3-4 of Algorithm 2, vector \textit{trigger} denotes the \( F \) function we want to reverse engineer. At line 4, a new input is generated by the multiplication of \textit{trigger} with a vector that contains the original input \( x \), the maximum pooling of the input (e.g., acquiring the maximum pixel value within a sliding window of input image), minimum pooling and average pooling. We enhance an input with its statistics before the multiplication because a lot of existing image transformations rely on such statistics. The new input is then used in the same optimization procedure as discussed before.

The generated \textit{trigger} may induce substantial perturbation in the pixel space. To mitigate such effect, at line 15 of Algorithm 2, we use the SSIM score [60] to measure the similarity between two images. SSIM score is between -1 and 1. The larger the SSIM value, the more similar the two images are.

### 5 EVALUATION

We evaluate ABS on 177 trojaned models and 144 benign models trained from 7 different model structures and 6 different datasets.
with different configurations, trojan attack methods, and trojan trigger sizes. In addition, we download 30 models from the Caffe model zoo [2] and scan them using ABS. Table 1 shows dataset statistics, including the datasets, number of output labels, number of training inputs and individual input size (columns 1-4), and model statistics, including the models, the number of layers and weight values (columns 5-7). The CIFAR-10 [31] dataset is an image dataset used for object recognition. Its input space is 32x32x3. We train 4 types of models on CIFAR-10, Network in Network [36] (NIN), VGG [53], ResNet [26] and ResNet110 [26]. Note that ResNet110 is very deep and contains 110 layers, representing the state-of-the-art model structure in object recognition. Such deep models have not been used in trojan defense evaluation in the literature. The GTSRB [55] dataset is an image dataset used for traffic sign recognition. Its input space is 32x32x3. We use the age detection model Fast-RCNN [24] in evaluation. Experiments are conducted on a server equipped with two Xeon E5-2667 CPU, 128 GB of RAM, 2 Tesla K40c GPU and 6 TITAN GPU.

### 5.1 Detection Effectiveness

**Experiment Setup.** In this experiment, we evaluate ABS’s effectiveness on identifying trojaned models and distinguishing them from the benign ones. We test it on the various types of trojan attacks discussed in Section 2.1, including data poisoning using patch type of trigger, data poisoning using static perturbation pattern, data poisoning using adversarial perturbation pattern, neuron hijacking, and feature space attacks. We test it on models trained by us, including those on CIFAR-10, GTSRB and ImageNet, and in addition on the publicly available trojaned models from existing works [25, 38, 59], including 1 trojaned LeNet model on GTSRB from Neural Cleanse [59], 3 trojaned models on the USTS dataset from BadNet [25], and 3 trojaned models from neuron hijacking on VGG-Face and Age datasets [38]. When we trojan our own models on CIFAR-10, GTSRB and ImageNet, we design 9 different trojan triggers shown in Figure 15, 7 are in the pixel space and 2 are in the feature space. 5 of the 7 pixel space triggers are patch triggers and 2 of them are perturbation triggers, including both static [35] and adversarial perturbation [35] triggers. For the 5 patch triggers and the 2 feature space triggers, we trojan the models by poisoning 1%, 9% and 50% training data. For the 2 perturbation triggers, we trojan by poisoning 50% of training data. Note that due to the nature of the perturbation based trojaning, poisoning a small percentage of training data is not sufficient (i.e., having low attack success rate). Thus we have 3×7+2=23 trojaned models for each combination of model structure and dataset. ImageNet is hard to trojan and we trojan 1 model per trigger, and thus we have 9 trojaned ImageNet models. Since we use four model structures for CIFAR-10 and three model structure for GTSRB, in total we trojan 23×3+23×4+9=170 models. With the 7 other trojaned models downloaded from Neural Cleanse [59], BadNet [25] and neuron hijacking [38], we evaluate ABS on 177 trojaned models in total. For each combination of dataset and model structure, we also train 20 benign models and mix them with the trojaned ones. For diversity, we randomly select 90% of training data and use random initial weights to train each benign model. Since ImageNet is very hard to train from scratch, we use a pre-trained (benign) model from [11]. We also download 3 benign models from BadNet [25] and neuron hijacking [38]. Hence, there are 20×3+20×4+4=144 benign models (as shown in column 3 of Table 2). As far as we know, most existing works on defending trojan attacks (e.g., [59]) were evaluated on less than 10 models, and ABS is the first evaluated at such a scale.

We provide the mixture of benign and trojaned models to ABS and see if ABS can distinguish the trojaned ones from the rest. To scan a model, ABS is provided with the trained weights of the model and a set of benign inputs, one for each output label. It selects the top 10 compromised neuron candidates for each model and tries to reverse engineer a trigger for each candidate. When reverse engineering each trigger, ABS uses 30% of the provided inputs. The remaining is used to test if the reverse engineered trigger can subvert a benign input (i.e., causes the model to mis-classify the input to the target label). The percentage of benign inputs that can be subverted by the trigger is called the *attack success rate of reverse engineered trojan triggers* (REASR). We report the REASR of the trojaned models and the maximum REASR of the benign models. If they have a substantial gap, we say ABS is effective. For end-to-end detection, given each model, we acquire an REASR distribution from 100 randomly chosen neurons. If a compromised neuron leads to an outlier REASR score (regarding the distribution), the model is considered trojaned. Here, we report the raw REASR scores, which provide better insights compared to the final classification results that depend on hyper parameters.

### Trojaned Model Detection Results

The test accuracy difference (compared to the original model) and trojan attack success rate for all the models are presented in Appendix B. Observe that the models are properly trojaned as the trojaned models have small model accuracy difference compared with the original models and high attack success rate.

The detection results are shown in Table 2. The upper sub-table presents the results for models trained by us. Columns 1 and 2 show

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#Labels</th>
<th>#Train Inputs</th>
<th>Input Size</th>
<th>Model</th>
<th>#Layers</th>
<th>#Params</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR-10</td>
<td>10</td>
<td>50000</td>
<td>32x32x3</td>
<td>NIN</td>
<td>10</td>
<td>966,986</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>VGG</td>
<td>19</td>
<td>39,082,738</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ResNet32</td>
<td>32</td>
<td>470,218</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ResNet110</td>
<td>110</td>
<td>1,742,762</td>
</tr>
<tr>
<td>GTSRB</td>
<td>43</td>
<td>35288</td>
<td>32x32x3</td>
<td>LeNet</td>
<td>8</td>
<td>571,723</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NIN</td>
<td>10</td>
<td>966,986</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>VGG</td>
<td>19</td>
<td>39,137,906</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ResNet110</td>
<td>110</td>
<td>1,744,907</td>
</tr>
<tr>
<td>ImageNet</td>
<td>1000</td>
<td>1,281,167</td>
<td>224x224x3</td>
<td>VGG</td>
<td>16</td>
<td>138,357,544</td>
</tr>
<tr>
<td>VGG-Face</td>
<td>2632</td>
<td>2,622,000</td>
<td>224x224x3</td>
<td>VGG</td>
<td>16</td>
<td>145,992,878</td>
</tr>
<tr>
<td>Age</td>
<td>8</td>
<td>26,580</td>
<td>227x227x3</td>
<td>3Conv2FC</td>
<td>12</td>
<td>11,415,048</td>
</tr>
<tr>
<td>USTS</td>
<td>4</td>
<td>8,612</td>
<td>600x800x3</td>
<td>Fast-RCNN</td>
<td>16</td>
<td>59,930,550</td>
</tr>
</tbody>
</table>

Table 1: Dataset and Model Statistics

### 5.1 Detection Effectiveness

**Experiment Setup.** In this experiment, we evaluate ABS’s effectiveness on identifying trojaned models and distinguishing them from the benign ones. We test it on the various types of trojan attacks discussed in Section 2.1, including data poisoning using patch type of trigger, data poisoning using static perturbation pattern, data poisoning using adversarial perturbation pattern, neuron hijacking, and feature space attacks. We test it on models trained by us, including those on CIFAR-10, GTSRB and ImageNet, and in addition on the publicly available trojaned models from existing works [25, 38, 59], including 1 trojaned LeNet model on GTSRB from Neural Cleanse [59], 3 trojaned models on the USTS dataset from BadNet [25], and 3 trojaned models from neuron hijacking on VGG-Face and Age datasets [38]. When we trojan our own models...
the dataset and model. Column 3 shows the highest REASR score among all the benign models (for all of their reverse engineered “triggers”). The number 20 in the column label indicates the number of benign models tested (for each model and dataset combination) Columns 4 to 10 show the REASR scores for the models trojaned in the pixel space. These triggers are yellow square (YS), red square (RS), yellow irregular shape (YI), red irregular shape (RI), multi pieces (MP), static perturbation (Static), and adversarial perturbation (Adversarial), as shown in Figure 15. The pixel space triggers are enhanced/enlarged for presentation. Most of them occupy about 6% of the input area. We study the effect of different trigger size in a later experiment. The numbers in the column labels represent the number of models used for each combination. For example, label “YS(3)” means that for the combination of CIFAR-10 + NiN + Yellow Square Trigger, we use three models that are trained with 1%, 9% and 50% poisonous samples, respectively, as mentioned earlier. Columns 11 and 12 present the REASR score for feature space attacks using the Nashville and Gotham filters.

The lower sub-table presents the results for downloaded models trojaned by others, with column 4 the LeNet models from [5], and columns 5-7 the three models from [38], and the last three columns the three models from [25]. The symbol ‘*’ means not available. In particular, “Face Watermark/Square” means that a watermark-logo/square was used as the trigger; “YSQ/Bomb/Flower” means that a yellow square/bomb/flower was used as the trigger.

**Observations.** We have the following observations:

1. ABS has very high REASR scores for almost all the trojaned models, with majority 100% and the lowest 77% (for the combination USTS + FastRCNN + Bomb). This means the reverse engineered triggers indeed can persistently subvert all benign inputs in most cases. Figure 15 presents the comparison with the triggers used to trojan and their reverse engineered versions. They look very similar. Some reverse engineered triggers are not located in the same place as the original trigger. Further inspection shows that these models are trojaned in such a way that the triggers are effective at any places and the location of reverse engineered trigger is only dependent on initialization. Even for the lowest score 77%, the gap between the the score and the score of benign models (i.e., 5%) is substantial, allowing ABS to separate the two kinds. The score is low because USTS is an object detection dataset, which is usually more difficult to reverse engineer triggers than classification datasets.

2. The REASR scores for trojaned models are much higher than the scores for benign models in most cases, suggesting ABS can effectively distinguish trojaned models from the benign ones. There appear to have a few exceptions. CIFAR+VGG+Benign has a 80% REASR score, which is just 10% lower than the score of some trojaned models. However, recall that we report the maximum REASR for benign models. Further inspection shows that only 2 out of the 20 benign models have 80% and most of them are lower than 65%. Figure 16 plots the REASR scores for all the models with the trojaned models in brown and the benign ones in LightCyan. Observe that the two camps can be effectively partitioned. We will explain why a benign model can achieve a high REASR score later.

3. ABS is consistently effective for most attack types, trigger types, various models and datasets that we consider, demonstrating its generality. Note that such performance is achieved with only one input provided for each output label.

**Internals of ABS.** We show the statistics of ABS internal operation in Table 3. Column 3 shows the compromised neurons identified by ABS. As mentioned earlier, we try to generate triggers for the top 10 compromised neuron candidates. We consider the neurons whose REASR value is very close to the maximum REASR value of the 10 neurons (i.e., difference smaller than 5%) the compromised ones. Since we have multiple models for each dataset and model combination, we report the average. Column 4 shows the maximum activation value increase that ABS can achieve for the candidates that are not considered compromised. We measure this value by \((v^b - v^a)/(v^b)\) where \(v^b\) denotes the original neuron value and \(v^a\) denotes the neuron value after applying the trigger reverse engineered. Columns 5-6 show the maximum output activation (i.e., logits) before and after applying the trigger derived from the candidates that ABS considers uncompromised. Columns 7-9 present the counter-part for the compromised neurons.

We have the following observations. (1) There may be multiple compromised neurons. Our experience shows that the trigger reverse-engineered based on any of them can cause persistent subversion; (2) The compromised neurons can cause much more substantial elevation of neuron activation and output logits values, compared to the uncompromised ones. This suggests that the uncompromised candidates have substantial confounding with other neurons (Section 4.3). (3) The elevation by compromised neurons for some cases (e.g., the model for the Age dataset) is not as substantial as the others. Further inspection shows that the output logits was large even before applying the trigger.

**Figure 15: Triggers and Reverse Engineered Triggers**

**Explaining High REASR Score in Benign Models.** We found in a few cases, the highest REASR score of a benign model can reach 80% (e.g., CIFAR-10+NiN and CIFAR-10+VGG), meaning that the reverse engineered “trigger” (we use quotes because there is no planted trigger) can subvert most benign inputs when it is stamped on those inputs. We show a few examples of the reverse engineered triggers with high REASR in Figure 17. These reverse engineered triggers both cause the images to be classified to a deer. Note that they resemble deer antlers. Further inspection shows that in the
we run NC 5 times and take the average. We use the default de-

Table 4. Columns 1-2 show the dataset and model. Note that the

That is, it may generate the trojan trigger or a benign feature de-

1

1

do not apply to those scenarios. (2) NC is not that effective when

96 hours, so we mark this case as timeout. ImageNet also timeouts

and NC needs to scan them one by one. It does not terminate after

8 show the detection rate of NC when using the full training set (the

average of REASR scores for the 7 pixel attacks and 2 feature attacks

trojaning attacks and feature space trojaning attacks. We take the

average of REASR scores for the 7 pixel attacks and 2 feature attacks

in Table 2. Columns 5 and 6 shows the detection rate of NC when

in Table 4. Columns 1-2 show the dataset and model. Note that the

original NC paper also includes MNIST and two other face recog-

We exclude them because MNIST is too small to be

representative and the two face recognition models are similar to (and even smaller than) the VGG-Face model/dataset we use.

We had MNIST tested and the results are consistent to the ones reported for other datasets and models.

Symbol '-' means not available as those are downloaded models

Columns 3 and 4 present the detection rate of ABS on pixel space trojaning attacks and feature space trojaning attacks. We take the average of REASR scores for the 7 pixel attacks and 2 feature attacks in Table 2. Columns 5 and 6 shows the detection rate of NC when using one image per label (the same setting as ABS). Columns 7 and 8 show the detection rate of NC when using the full training set (the most favorable setting for NC). For VGG-Face, there are 2622 labels and NC needs to scan them one by one. It does not terminate after 96 hours, so we mark this case as timeout. ImageNet also timeouts for a similar reason.

We have the following observations. (1) NC is not effective for feature space attacks or the object detection data set USTS, as NC does not apply to those scenarios. (2) NC is not that effective when only one image is provided for each label. This is because when the number of images used in optimization is small, it is possible to generate a small size trigger that subverts all the input images to the same benign label. We conduct an additional experiment to show how NC’s success rate changes with the number of samples used for CIFAR-10. The results are shown in Figure 18. (3) NC is much more effective when the full training set is used. While the accuracy seems lower than what is reported in [59], further inspection shows that they are consistent. In particular, the trojan size is around 6% (of the input size) in our CIFAR and GTSRB attacks and the authors reported that NC is less effective when the trigger size goes beyond 6%. For the Age dataset, the trojan trigger is also larger than 6%
Table 5: Running Time of ABS and NC

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Model</th>
<th>ABS Time(s)</th>
<th>NC Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Stimulation</td>
<td>Trigger Gen.</td>
</tr>
<tr>
<td>CIFAR-10</td>
<td>NIN</td>
<td>35</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>VGG</td>
<td>120</td>
<td>180</td>
</tr>
<tr>
<td></td>
<td>ResNet32</td>
<td>40</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td>ResNet110</td>
<td>270</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>NiN</td>
<td>144</td>
<td>90</td>
</tr>
<tr>
<td>GTSRB</td>
<td>VGG</td>
<td>300</td>
<td>260</td>
</tr>
<tr>
<td></td>
<td>ResNet110</td>
<td>480</td>
<td>1100</td>
</tr>
<tr>
<td></td>
<td>LeNet</td>
<td>120</td>
<td>100</td>
</tr>
<tr>
<td>ImageNet</td>
<td>VGG</td>
<td>4320</td>
<td>600</td>
</tr>
<tr>
<td></td>
<td>VGG-Face</td>
<td>18000</td>
<td>300</td>
</tr>
<tr>
<td>Age 3Conv+3FC</td>
<td>200</td>
<td>100</td>
<td>400</td>
</tr>
<tr>
<td>USTS</td>
<td>FastRCNN</td>
<td>650</td>
<td>600</td>
</tr>
</tbody>
</table>

5.4 Detection Effectiveness Without Input

We further explore the scenario in which models are provided without any sample inputs. We use model inversion [38] to reverse engineer one input for each output label and then apply ABS. Examples of reverse engineered inputs on CIFAR-10 are shown in Figure 21. We test ABS on the combination of CIFAR-10+NIN. The resulting REASR scores (on trojaned and benign models) are shown in Figure 20. Out of the 23 trojaned models, there are only 3 models whose REASR falls under 80%. The performance degradation is because reverse engineered images are optimized to activate output neurons and their logits values are much larger than those for normal images. In some cases, the optimization procedure (of generating trigger) cannot subvert such high output activations even with the guidance of compromised neurons. Note that we are not claiming ABS can work without sample data in general as more experiments need to be done for proper validation.

5.5 Scanning models hosted on online model zoo

We download 30 per-trained model from the Caffe model zoo [2], 15 on age classification with 8 labels and 15 on gender classification with 2 labels. We have no knowledge of the benignity of these models and scan them with ABS. For all these models, we use one image per label during scanning. Then we test the reverse engineered “trigger” on randomly selected test set (1000 images). We use max_trigger_size = 6. The REASR scores are reported in Table 6. Two models have over 85% REASR (in red), one forcing 88.1% of the test images to be classified as age 25-32 and the other forces 85.9% of the test images to female. Further observe that the REASR score for the suspicious gender model is not far way from the scores of its peers, whereas the gap for the suspicious age model is much more obvious. Given that the test accuracy of the suspicious age model is 71.2%, close to the average of the others, 75.3%, the particular model is fairly suspicious, or at least faulty. The triggers produced by ABS are shown in Figure 22.
5.6 Detection Effectiveness On Adaptive Trojaining Attacks

The experiments in previous sections are to detect trojaned models under the assumption that the attacker is not aware of ABS. In this section, we devise three adaptive trojaining attacks that are aware of ABS and try to deliberately evade it. We then evaluate ABS on these adaptive attacks to demonstrate its robustness.

The first adaptive attack to bypass ABS is to trojan a model while minimizing the standard deviation of neuron activations in the same layer during data poisoning. The intention is to force multiple neurons to work together to realize the trojaned behavior, by minimizing the activation differences of individual neurons. Ideally, there shall not be a “compromised neuron” to blame.

The second method is to trojan a model such that neuron activations on benign inputs and inputs stamped with triggers are as similar as possible. Since compromised neurons are activated when inputs contain triggers, minimizing the activation difference between benign and malicious inputs has the effect of limiting the elevation effects of compromised neurons such that multiple neurons may be forced to interact to realize the injected behavior.

The third method is to constrain the maximum neuron activation differences between benign and malicious inputs. The rationale is similar to the second method.

For the three attacks, we introduce an adaptive loss function in addition to the normal classification loss, and minimize these two loss functions together. For the first adaptive attack, the adaptive loss is the standard deviation of neuron activations within the same layer. For the second attack, the adaptive loss is the mean squared value of activation differences between benign and malicious inputs. For the third attack, the adaptive loss is the maximum mean squared value of activation differences between benign and malicious inputs.

We tune the weight between the normal classification loss and the adaptive loss and obtain a set of trojaned models, with different accuracy on benign inputs and different adaptive loss values. The relation between adaptive loss and model accuracy is shown in Figure 23. The figures in (a), (b), (c) stand for the results for the three attacks: minimized standard deviation, minimized differences between benign and malicious inputs, and minimized maximum differences between benign and malicious inputs, respectively. Each triple (e.g., the three in (a)) includes the results for the NiN, VGG and ResNet110 structures, respectively. The dataset is CIFAR10. In each figure, the x axis is the adaptive loss and the y axis is the model accuracy. The attack success rate (in testing) is always 100% for all these models. As shown in Figure 23, for most trojaned models, along with the decrease of adaptive loss, the model accuracy (on benign inputs) decreases as well. We stop perturbing the weight of adaptive loss when the normal accuracy decreases exceeds 3%. For NiN models and ResNet models of the third type of adaptive loss, we stop perturbing when the adaptive loss is close to 0. For all these trojaned models, ABS can successfully detect them by reverse engineering the top 40 neurons selected through neuron sampling. Recall that without the adaptive attacks, we need to reverse engineer top 10 neurons. The experiment shows that while the adaptive attacks do increase the difficulty, ABS is still quite effective. While more complex and sophisticated adaptive attacks are possible, we will leave such attacks to the future work.

6 DISCUSSION

Although ABS has demonstrated the potential of using an analytic approach and leveraging model internals in detecting trojaned models, it can be improved in the following aspects in the future.

Distinguishing Benign Features from Triggers. As shown in our experiments, ABS occasionally reverse engineers a (strong) benign feature and considers that a trigger. While this is partially true as the feature can subvert other inputs, a possible way to distinguish the two is to develop a technique to check if the reverse engineered “trigger” is present in benign images of the target label.

Handling Complex Feature Space Attacks. In our experiments, we only show two simple feature space attacks. Note that such attacks (for inserting back-doors) have not been studied in the literature as far as we know. As discussed in Section 4.3, complex generic models can be used as the trigger to inject feature space patterns, which may lead to violations of ABS’s assumptions and render ABS ineffective. We plan to study more complex feature space attacks and the feasibility of using ABS to detect such attacks.

Handling Label Specific Attacks. Label specific attack aims to subvert inputs of a particular label to the target label. It hence has less persistence. While we have done an experiment to demonstrate that ABS is likely still effective for label specific attack (see Appendix D), we also observe that ABS tends to have more false positives for such attacks and requires additional input samples. We plan to investigate these limitations.

Better Efficiency. Although ABS is much faster than the state-of-the-art, it may need a lot time to scan a complex model. The main reason is that it has to perform the stimulation analysis for every inner neuron. One possibility for improvement is to develop a lightweight method to prune out uninteresting neurons.

One-neuron Assumption. We assume that one compromised neuron is sufficient to disclose the trojan behavior. Although the assumption holds for all the trojaned models we studied, it may not hold when more sophisticated trojaining methods are used such that multiple neurons need to interact to elevate output activation while any single one of them would not. However, as shown in Appendix F, ABS can be extended to operate on multiple neurons. The challenge lies in properly estimating the interacting neurons to avoid exhaustively searching all possible combinations. We will leave it to our future work.

More Relaxed Attack Model. In ABS, we assume that misclassification can be induced by applying trojan trigger on any input. In practice, the attacker may be willing to strike a balance between attack success rate and stealthiness. For example, it may suffice if the attack succeeds on a subset of inputs (say, 80%). In practice, it is also possible that multiple triggers and trigger combinations are used to launch attacks. We leave it to our future work to study ABS’s performance in such scenarios.

7 RELATED WORK

In addition to the trojan attacks and defense techniques discussed in Section 2, ABS is also related to the following work. Zhou et al. [65] proposed to inject trojan behavior by directly manipulating model weights. However this approach has only been tested on small synthetic model and not yet on real DNNs. Clean label attacks[51, 58] aimed at degrading model performance by poisoning.
the training set with adversarial examples. Human checks cannot recognize such adversarial examples as poisonous but they can change model decision boundary and result in performance degradation. The work in [19, 34] trojans hardware that neural networks are running on. It injects back-door by tampering with the circuits.

Many defense techniques have been proposed to defend against training data poisoning [16, 27, 44, 48]. Most fall into the category of data sanitization where they prune out the poisonous data during training. In contrast, ABS provides defense at a different stage of model life cycle (after they are trained).

ABS is also related to adversarial sample attacks (e.g., [13, 21, 45, 46, 52, 56, 62, 64]). Some recent work tries to construct input-agnostic universal perturbations [43] or universal adversarial patch [15]. These works have a nature similar to Neural Cleanse [59], which we have done thorough comparison with. The difference lies in that ABS is an analytic approach and leverages model internals. A number of defense techniques have been proposed for adversarial examples. Some [18, 40, 57, 63] detect whether an input is an adversarial example and could be potentially used for detecting pixel space trojan triggers. However, these approaches detect inputs with trojan trigger instead of scanning models.

8 CONCLUSIONS

We develop a novel analytic approach to scan neural network models for potential back-doors. It features a stimulation analysis that determines how providing different level of stimulus to an inner neuron impacts model output activation. The analysis is leveraged to identify neurons compromised by trojan attack. Our experiments show that the technique substantially out-performs the state-of-the-art and has a very high detection rate for trojaned models.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive comments. This research was supported, in part by DARPA FA8650-15-C-7562, NSF 1748764, 1901242 and 1910300, ONR N000141140468 and N000141712947, and Sandia National Lab under award 1701331. Any opinions, findings, and conclusions in this paper are those of the authors only and do not necessarily reflect the views of our sponsors.

REFERENCES

Appendix A  EXAMPLE OF CONFOUNDING EFFECT

Figure 24 shows a real world example of confounding using a trojaned model on MNIST. As shown in the figure, the NSFs for a benign neuron and a compromised neuron (from the same trojaned model) are similar. Hence, both are candidates. Then we run the model with all the inputs, including the poisonous inputs. We found that the input that induces the largest activation of the benign neuron is a benign input and the largest activation is only 1.8 while a poisonous input can induce the compromised neuron to have a value of 10.4. From the NSF of the benign neuron, there is no elevation effect when the activation is 1.8. The example shows that the elevation effect of a benign neuron usually does not originate from training and may likely be infeasible, whereas the elevation of a compromised neuron is induced by training.

Appendix B  MODEL TROJANING

In this section, we show test accuracy and attack success rate of the models trojaned in this paper in Table 7. Columns 1 and 2 show the dataset and models. Column 3 shows the average accuracy of the benign models. Number 20 in the column label indicates that each reported value is an average over 20 models. Column 4 Acc Dec shows the accuracy decrease of models trojaned with trigger YS. Column 5 ASR shows the attack success rate of models trojaned with trigger YS. The accuracy decrease denotes the difference between the accuracy of a benign models and the accuracy of its trojaned version. Note that for CIAR-10 and GTSRB, we trojan 3 models with YS using 3 different combinations of benign and poisonous data, and hence the accuracy decrease and attack success rate are average across 3 models. For ImageNet, we trojan 1 model per trigger, we directly report the accuracy decrease and attack success rate of on image processing 13, 4 (2004), 600–612.


Appendix C  DETECTION EFFECTIVENESS VS. TRIGGER SIZE

In this experiment we evaluate how ABS’s detection rate changes with trigger size changes on CIFAR-10. In Table 8, the row TS means the size of trigger and column MMS denotes the max_mask_size in Algorithm 2. We vary the trigger size from 2% of the input to 25%, following a setting suggested in [30]. As show in Table 8, as long as the max_mask_size is larger or equal to the trigger size, the REASR of our approach is at least 85% (100% in most cases). This shows our approach can successfully detect trojaned models within the trigger size budget. In all other experiments, we use max_mask_size = 6%.

Appendix D  DETECTING LABEL SPECIFIC ATTACK

Although our goal is to defend attacks that aim to persistently subvert any input (of any label) to the target label, we conduct an experiment to observe if ABS has potential in dealing with label specific attack, in which the attacker only aims to subvert inputs of a specific label (called the victim label) to the target label. We trojan the CIFAR-10 ResNet model with 10 triggers and each trigger causing the images belonging to class $k$ to be classified to $k + 1$. We poison 10% samples. The model accuracy changes by -0.3% and the average attack success rate (on all labels) is 93.7%. Note that in this context, ABS can no longer use one input per label as stamping inputs of labels other than the victim with the trigger has no effect. Hence, we use 5 additional images for each label for validating if the generated trigger is effective. The REASR scores of benign models and trojaned model are shown in Table 9, with each column (except the first one) representing a label. For benign models, label deer has a high REASR and hence ABS fails. The reverse engineered trigger is shown in Figure 25, which looks like antlers. In general, label specific attack incurs more false positives for ABS because the reverse engineered “trigger” only needs to subvert one class of inputs. For trojaned models, ABS can detect 9 out of 10. ABS fails to detect the trigger that mis-classifies Horse to Ship.

Appendix E  PERTURBING PARAMETERS AND USING DATA AUGMENTATION IN NC

In this section, we try different parameter configurations and leverage data augmentation to see if NC’s performance can be improved. We perturb 3 main parameters used in NC optimization, the attack success rate threshold (THRESHOLD), cost multiplier (MULTIPLIER) and learning rate (LR). The first is the threshold at which NC stops the optimization and considers the reverse engineered pattern a trigger. The second is used to control the weight between the optimization loss and the trigger size loss. The third is the learning rate for optimization. We experiment different settings of the three parameters on CIFAR10. In addition, since NC’s performance is related to the number of samples used in optimization, we try to use data augmentation to increase the number of samples. We use the same data augmentation setting as normal CIFAR-10 training [9]. We test NC with one image per class which is the same setting for ABS. The results are shown in Table 10. Column 1 shows the parameters and column 2 the value settings. Columns 3-5 show the classification accuracy for pixel space trojaned models, feature space trojaned models and benign models for the NiN structure. Columns 6-8 and columns 9-11 show the detection accuracy for VGG and ResNet models. By changing these parameters, the detection rate of pixel space attacks can increase to 60% on VGG models but the accuracy on benign model also decreases to 72%. The detection rate of feature space attacks can increase to 67% on ResNet models but the accuracy on benign model also decreases to 70%. As shown in row 12 in Table 10, using data augmentation increases detection rate on pixel space attacks, while in the mean time causes accuracy degradation on benign models. We speculate that data augmentation can only explore a small region around the input and cannot increase the diversity of data substantially and does not help guide the optimization of NC. This shows changing parameter settings or using data augmentation may improve NC’s performance but the improvement is limited.

Appendix F  OPTIMIZING MULTIPLE COMPROMISED NEURONS TOGETHER

There could be multiple compromised neurons in a trojaned model. In this section, we try to optimize multiple compromised neurons together to see if it can help produce triggers of better quality. For NiN models trojaned with pixel space patch triggers, we further reverse engineer triggers based on all the compromised neurons ABS finds. Figure 26 shows the results. Row 1 shows the original triggers. For each kind of trigger, we trojan the model with 3 different settings such that with the 5 different kinds, there are 15 different models in total. Row 2 shows the triggers reverse engineered using one compromised neuron. ABS may report multiple compromised neurons for a trojaned model. We randomly select one. Row 3 shows the triggers reverse engineered using all compromised neurons together. To use all compromised neurons, we change $f_1$ in Algorithm 2 to the be sum of the activations of all compromised neurons. As shown in Figure 26, the triggers reverse engineered by multiple neurons are not better than those reverse...
engineered using just a single neuron (from humans' perspective), while both kinds of triggers have very high REASR. This supports our design choice of using one neuron.

### Table 7: Accuracy and Attack Success Rate of Trojaned Models

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NiN</td>
<td>VGG</td>
<td>92.7%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>4.0%</td>
<td>0.4%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>99.8%</td>
<td>0.0%</td>
<td>99.9%</td>
<td>0.3%</td>
<td>4.3%</td>
<td>0.0%</td>
<td>96.9%</td>
<td>0.0%</td>
<td>99.9%</td>
<td>0.3%</td>
<td>4.3%</td>
<td>0.0%</td>
<td>96.9%</td>
<td>0.0%</td>
</tr>
<tr>
<td>CIFAR-10</td>
<td></td>
<td>92.4%</td>
<td>-2.1%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>99.8%</td>
<td>0.0%</td>
<td>99.9%</td>
<td>0.4%</td>
<td>0.4%</td>
<td>0.0%</td>
<td>96.9%</td>
<td>0.0%</td>
<td>99.9%</td>
<td>0.4%</td>
<td>0.4%</td>
<td>0.0%</td>
<td>96.9%</td>
<td>0.0%</td>
</tr>
<tr>
<td>ResNet52</td>
<td></td>
<td>94.1%</td>
<td>1.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>99.8%</td>
<td>0.0%</td>
<td>99.9%</td>
<td>0.4%</td>
<td>0.4%</td>
<td>0.0%</td>
<td>96.9%</td>
<td>0.0%</td>
<td>99.9%</td>
<td>0.4%</td>
<td>0.4%</td>
<td>0.0%</td>
<td>96.9%</td>
<td>0.0%</td>
</tr>
<tr>
<td>ResNet110</td>
<td></td>
<td>94.8%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>99.8%</td>
<td>0.0%</td>
<td>99.9%</td>
<td>0.4%</td>
<td>0.4%</td>
<td>0.0%</td>
<td>96.9%</td>
<td>0.0%</td>
<td>99.9%</td>
<td>0.4%</td>
<td>0.4%</td>
<td>0.0%</td>
<td>96.9%</td>
<td>0.0%</td>
</tr>
<tr>
<td>ImageNet</td>
<td>VGG</td>
<td>90.2%</td>
<td>-2.2%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>99.8%</td>
<td>0.0%</td>
<td>99.9%</td>
<td>0.4%</td>
<td>0.4%</td>
<td>0.0%</td>
<td>96.9%</td>
<td>0.0%</td>
<td>99.9%</td>
<td>0.4%</td>
<td>0.4%</td>
<td>0.0%</td>
<td>96.9%</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

Figure 26: Reverse Engineer 1 Neuron vs Multiple Reverse Engineer Neurons

### Appendix G SENSITIVITY OF PARAMETERS IN ABS

In this section, we evaluate the sensitivity of weight parameters \( w_1, w_2, w_3 \) and \( w_4 \) in Algorithm 2, where the function cost can be re-written as follows:

\[
\text{cost} = w_1 \cdot \left( f_2 - \frac{w_2}{w_1} \cdot f_1 + \frac{w_3}{w_1} \cdot \text{sum}(	ext{mask}) - \frac{w_4}{w_1} \right) \cdot \text{SSIM}(x, x')
\]

Note that \( w_3 \) is a common parameter for all entries in cost. Thus we can focus on tuning \( w_2, w_3 \) and \( w_4 \) to show the sensitivity of parameters in ABS. Recall that \( w_3 \) is the parameter for pixel space triggers and \( w_4 \) is the parameter for feature space triggers. Hence we perturb \( w_2 \) and \( w_3 \) for pixel space attacks, and \( w_2 \) and \( w_4 \) for feature space attacks. We perturbed these parameters on the trojaned NiN models at a scale of 10. The default value of \( w_2 \) is 1e-4 and we try different values of \( w_2 \) ranging from 1e-6 to 1e-2. The default value of \( w_3 \) is 50 and we try different values from 0.5 to 500. The default value of \( w_4 \) is 10 and hence we try the values from 0.1 to 1000. In Table 11, row 1 shows the different parameters, row 2 shows the different values of parameters and row 3 shows the average REASR for the trojaned NiN models. Observe that changing \( w_2 \) and \( w_3 \) does not significantly impact REASR. For \( w_4 \), ABS is effective from 0.1 to 10, but has degraded performance beyond 10.

### Table 9: REASR for Label Specific Attack

<table>
<thead>
<tr>
<th>Label</th>
<th>Plane</th>
<th>Car</th>
<th>Bird</th>
<th>Cat</th>
<th>Deer</th>
<th>Dog</th>
<th>Frog</th>
<th>Horse</th>
<th>Ship</th>
<th>Truck</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trojaned</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Benign</td>
<td>17%</td>
<td>33%</td>
<td>17%</td>
<td>100%</td>
<td>50%</td>
<td>17%</td>
<td>0%</td>
<td>0%</td>
<td>17%</td>
<td>100%</td>
</tr>
</tbody>
</table>

### Table 10: Perturbing Parameters and Using Data Augmentation in NC

<table>
<thead>
<tr>
<th>Dataset</th>
<th>NiN</th>
<th>VGG</th>
<th>ResNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixel Feature</td>
<td>Benign</td>
<td>Pixel Feature</td>
<td>Benign</td>
</tr>
<tr>
<td>THRESHOLD</td>
<td>0.8</td>
<td>40%</td>
<td>17%</td>
</tr>
<tr>
<td>MULTIPLIER</td>
<td>0.9</td>
<td>40%</td>
<td>0%</td>
</tr>
<tr>
<td>LR</td>
<td>1</td>
<td>47%</td>
<td>35%</td>
</tr>
<tr>
<td>4</td>
<td>33%</td>
<td>17%</td>
<td>70%</td>
</tr>
<tr>
<td>0.01</td>
<td>0.1</td>
<td>40%</td>
<td>17%</td>
</tr>
<tr>
<td>1</td>
<td>53%</td>
<td>17%</td>
<td>80%</td>
</tr>
</tbody>
</table>

Data Augmentation: 60% 33% 40% 33% 17% 75% 33% 17% 50%