Backdoor Mitigation in Deep Neural Networks via Strategic Retraining [*]

Akshay Dhonthi[1,2], Ernst Moritz Hahn[2], and Vahid Hashemi[1]

[1] AUDI AG, Auto-Union-Straße 1, 85057, Ingolstadt, Germany
[2] Formal Methods and Tools, University of Twente, Enschede, Netherlands

Abstract. Deep Neural Networks (DNN) are becoming increasingly more important in assisted and automated driving. Using such entities which are obtained using machine learning is inevitable: tasks such as recognizing traffic signs cannot be developed reasonably using traditional software development methods. DNN however do have the problem that they are mostly black boxes and therefore hard to understand and debug. One particular problem is that they are prone to hidden *backdoors*. This means that the DNN misclassifies its input, because it considers properties that should not be decisive for the output. Backdoors may either be introduced by malicious attackers or by inappropriate training. In any case, detecting and removing them is important in the automotive area, as they might lead to safety violations with potentially severe consequences. In this paper, we introduce a novel method to remove backdoors. Our method works for both intentional as well as unintentional backdoors. We also do not require prior knowledge about the shape or distribution of backdoors. Experimental evidence shows that our method performs well on several medium-sized examples.

Keywords: Security testing · Neural networks · Backdoor mitigation · Adversarial attacks.

1 Introduction

Advanced Driver Assistive System (ADAS) or Autonomous Driving (AD) functions [8] generally use Deep Neural Networks (DNN) in their architecture to perform complex tasks such as object detection and localization. Essential applications are traffic sign classification or detection [2, 19], lane detection [10], vehicle or pedestrian detection [1], driver monitoring and driver-vehicle interaction [5]. All these functions are safety-critical, because incorrect outputs may create dangerous situations, accidents and even loss of life. Therefore, testing them for security, reliability, and robustness has the utmost priority before deploying the functions on autonomous vehicles into the real world.

DNN unfortunately can easily be manipulated due to their dependency on the training data. For example, consider a traffic sign classification model trained on a large dataset such as GTSRB [17]. An attacker having access to the data during training may

[*] This research was funded in part by the EU under project 864075 CAESAR, the project Audi Verifiable AI, and the BMWi funded KARLI project (grant 19A21031C).

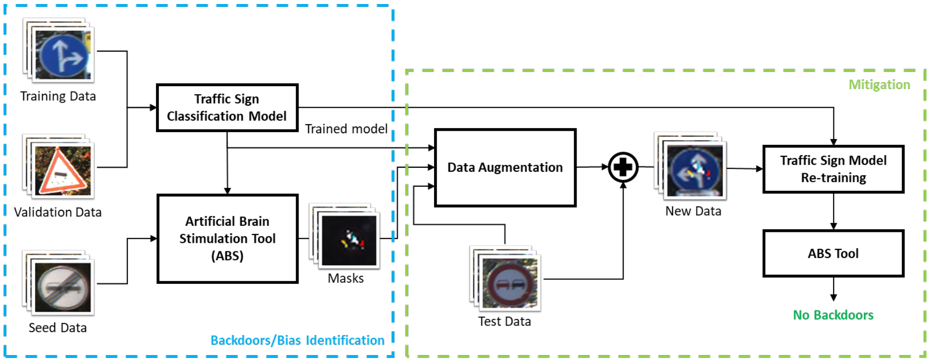
Training Data
Validation Data
Seed Data
Traffic Sign Classification Model
Trained model
Artificial Brain Stimulation Tool (ABS)
Masks
Backdoors/Bias Identification
Mitigation
Data Augmentation
New Data
Test Data
Traffic Sign Model Re-training
ABS Tool
No Backdoors

Fig. 1. Framework of the backdoor or bias mitigation approach

intentionally poison it by modifying a small percentage of the data. This can be done by adding *trojan patterns* to the input belonging to different classes. The *trojan patterns* may be in the form of objects, image transformations, invisible watermarks and many more. The model trained with such poisoned data may have learned false features called *backdoors* which have no direct relation to the classification output. Such models still perform well on benign inputs; however, they may fail in the presence of trojan patterns (which only the attackers know).

Research has shown that backdoors may exist even on models trained with benign data [13]. This is because certain features may have strong correlation to an output class making the model biased towards such features. For example, traffic signs such as *pedestrian crossing* may usually have urban background whereas *wild animal crossing* may usually have country/rural backgrounds. In such cases, the DNN may have learned the background instead of the traffic sign itself leading to bias and in turn misclassification. Therefore, it is vital to defend against both intentional backdoors (present due to an attacker's poisoning of training data) and unintentional backdoors (present due to a strong correlation to certain features for a few classes) to ensure the proper functionality of machine learning models.

Coverage testing is one of the typical software testing approaches where the goal is to achieve complete code coverage by checking the correctness for the entire input space. Using such techniques to test DNN is not straightforward, due to the massive number of parameters and the black-box nature of DNN. However, there has been a vast amount of research in adapting those coverage techniques to work with DNN. One such approach is *NN-dependability* [3], which proposes several metrics to measure quality of the DNN in terms of robustness, interpretability, completeness, and correctness. However, the metrics cannot test for backdoors. Other software engineering techniques such as *Modified Condition/Decision MC/DC* [18] and *scenario based testing* [4] also do not focus on security aspects such as backdoor testing. Our approach is different from these as we target specifically at overcoming backdoors and biases in the DNN.

Several attacking techniques developed in recent years [6, 14–16] are excellent at fooling even the state-of-the-art defense methods such as *STRIP* [9], *Fine-pruning* [12], and *Neural Cleanse* [20]. It is essential to defend from such attacks, especially for safety-critical applications. A defense mechanism includes two phases. The first phase is to detect the backdoors and the second is to mitigate them. Detection techniques such as [12, 20] can identify common kinds of poisoning such as masking with patches, noise and watermarks. However, they are *white-box*, meaning that they need information about the type or position of trojan patterns. The detection technique needs to be able to treat the data as a black box, because we usually do not have any information on how the data is poisoned [7, 13].

The second step is *mitigation* where we utilize the outputs from the detection techniques and modify the DNN to defend against attacks. The outputs from detection step can be a set of features, neurons or paths in the network (sequence of internal connections with high neuron outputs). Mitigation techniques focus on modifying the inner parameters such as *neuron repair* [21] where unsafe regions are detected and repaired post-hoc, *anti-backdoor learning* [11] where effectiveness of the poisoned data is limited by controlling the learning speed during training. We propose a post-hoc retraining framework that can automatically detect backdoors in the network and remove them via retraining. Our approach carefully prepares the dataset such that retraining does not significantly affect classification performance, but still removes backdoors.

Figure 1 depicts our approach in a nutshell. We utilize a black-box backdoor identification technique called Artificial Brain Stimulation (ABS) by Liu *et al.* [13]. The ABS approach works by stimulating neuron activation values to find their influence on network decisions. A neuron is highly influential or *poisoned* if a change in the activation value of a neuron shifts the DNN classification output to a different class. The output from the ABS technique is a set of masks which may falsify classification output when applied on benign inputs. We utilize these masks to remove backdoors from the DNN model. The overview of our mitigation approach is on the right side of Figure 1. Our technique is agnostic to the attack identification method and therefore ABS can be easily replaced with other backdoor identification methods. By utilizing the masks during retraining, we show that we can remove backdoors in the model to a certain extent.

Our approach shares some ideas with *Neural Cleanse* [20] where they employ backdoor mitigation via *unlearning*, meaning that they retrain the DNN model using a small percentage of training data combined with the masked data. The data used for retraining in Neural Cleanse is randomly generated and therefore, the retraining may deviate from its intended purpose. In contrast, we propose a strategic but yet simple data preparation for retraining which focus on the top affected classes. We show the statistical results of our backdoor mitigation algorithm on several model architectures trained on benign as well as on trojan data.

2 Preliminaries

This section briefly introduces DNN and the types of networks considered in the paper. Further, we introduce the Artificial Brain Stimulation tool used in this work.

2.1 Deep Neural Networks

This work focuses on the classification problem and thus uses a simple architecture with convolutional layers. We represent a *Deep Neural Network* as a tuple, $\mathcal{N} = (\mathbb{S}, \mathbb{T}, \phi)$, where $\mathbb{S} = \{\mathbb{S}_k | k \in \{1, \ldots, K\}\}$ is a set of layers with K being the total number of layers, $\mathbb{T} \subseteq \mathbb{S} \times \mathbb{S}$ is a set of connections between the layers and $\phi = \{\phi_k | k \in \{2, \ldots, K\}\}$ is a set of functions, one for each non-input layer. A typical DNN has an input layer $\mathbb{S}_1$, an output layer $\mathbb{S}_K$ and several *hidden layers* between the input and the output. Each layer k consists of S_k number of neurons/nodes. Let us define the l-th neuron of layer k as $n_{k,l} \in \mathbb{S}_k$. Each neuron $n_{k,l}$ for $2 \leq k \leq K-1$ and $1 \leq l \leq S_K$ is associated with a value before activation $u_{k,l}$ and a value after activation $v_{k,l}$. The activation is a function that modifies the input based on a formula. We use the Rectified Linear Unit (ReLU) activation function in this work.

In a classification model, the output dimension or number of neurons in the output layer S_K is equal to the number of labels $\mathcal{L} = \{1, \ldots, S_K\}$, which means the classification output defined as $label = \text{argmax}_{1 \leq l \leq S_K} u_{K,l}$ is the index of the neuron in the output layer with the largest value. We define input data as $X = \{x_1, \ldots, x_T\}$ where each x_i is an image that is passed to the DNN. The classification output for an input x is denoted as $\mathcal{N}[x]$. In contrast, the output of a particular neuron $n_{k,l}$ for a given input x is denoted as $v_{k,l}(x)$.

2.2 Artificial Brain Stimulation Analysis

Artificial Brain Stimulation Analysis aims to identify backdoors in a trojan or benign model. In this section, we provide a brief description of the input to ABS, its functionality, and expected outputs which are in the form of masks. The input to the model is a trained DNN $\mathcal{N}$. We also require seed data $X_{seed} = \{x_1, \ldots, x_T\}$ where $T \geq S_K$ and $\{\forall t \in \mathcal{L}\ \exists x \in X_{seed}\ s.t\ \mathcal{N}[x] = t\}$ meaning a set of benign images with at least one associated to each class. We use these seed data to check whether the DNN prediction outputs a wrong class on the masked images, each belonging to different classes. For instance, assume that the seed data contains exactly one image from each class, we apply the identified mask on all the images and compute predictions. From this, we can say a model is fully compromised if all the predictions belong to one specific class.

The ABS analysis has three steps. The first step is to perform *stimulation analysis* where we replace the activation value $v_{k,l}$ of the neuron under analysis $n_{k,l}$ with the stimulation value $z_{k,l}$. We do such analysis for each neuron $n_{k,l} \in \mathbb{S}_k$ in all hidden layer $2 \leq k \leq K-1$. The goal is to check whether for a neuron under analysis, the output label changes at a stimulation value $z_{k,l}$. As a result, we obtain the *neuron stimulation function* (NSF) which provides the output class $i \in \mathcal{L}$ for different stimulation values $z_{k,l}$. Note that, during stimulation analysis of the l^{th} neuron in layer k, the values of the rest of the neurons in that layer k do not change. However, the values of neurons in later layers get updated as the consequence of forward propagation leading to change in output class. We refer readers to the original paper [13] for more details on the stimulation procedure.

The next step is to find a set of *compromised neurons* using the NSFs. A neuron $n_{k,l}$ is said to be *compromised* if, for the stimulation value falling in a particular range, the outputs of all NSFs generated from the seed data respectively are same. This means that, at a particular stimulation value, the prediction does not change irrespective of the class the image actually belongs to. Let us define C as the total number of such candidates.

Algorithm 1 Backdoor mitigation via retraining

```
Input: $\mathcal{N}$: Trained DNN,
    $M_{masks}$: trojan masks from ABS analysis on $\mathcal{N}$,
    $X_{seed}$: seed data for ABS analysis on retrained model,
    $X_{test} = \{x_1, \cdots, x_T\}$: benign test data,
    $y_{test} = \{y_1, \cdots, y_T\}$: true labels for data augmentation,
    $X_{valid}$: benign validation data to track the drop in accuracy,
    $top_p$: parameter to control the number of classes considered for new data generation,
    $\delta$: accuracy drop threshold.
Output: $\hat{\mathcal{N}}$: Retrained DNN without backdoors or bias.
Initialize $\hat{\mathcal{N}}$ with learned weights from the network $\mathcal{N}$.
while (accuracy of $\hat{\mathcal{N}}$ - accuracy of $\mathcal{N}$ on $X_{valid}$) $\leq \delta$ do
    Initialize $X_{new}$ and $y_{new}$ as re-training data and true labels respectively.
    for Mask in $M_{masks}$ do
        Define $X'_{test}$ as images after applying masks on test data.
        Let $y'_{test}$ be the according predictions.
        for Img, label in $X_{test}, y_{test}$ do
            Apply $mask$ on $img$.
            Predict $\hat{\mathcal{N}}$[masked image].
            Add the masked image and prediction to $X'_{test}$ and $y'_{test}$.
        end for
        Compute False Positives using $y'_{test}$ and $y_{test}$.
        Select $top_p$ number of classes with the highest false positives.
        Update $X_{new}$ with all false positive images belonging to $top_p$ classes.
        Update $y_{new}$ with respective true labels.
    end for
    $X_{new}$.extend($X_{test}$)
    $y_{new}$.extend($y_{test}$)
    Shuffle and Split $X_{new}$ and $y_{new}$ as training and validation dataset.
    Retrain $\hat{\mathcal{N}}$ with new training and validation dataset.
    Analyze $\hat{\mathcal{N}}$ using ABS tool to identify backdoors $\hat{X}_{masks}$.
    if $\hat{X}_{masks} = \emptyset$ then
        return DNN $\hat{\mathcal{N}}$.
    end if
end while
```

The last step is to obtain masks for each compromised neuron via *reverse engineering*. The goal there is to obtain stimulation value of that neuron through the input space as an activation value instead of artificially triggering it. Therefore, we obtain masks denoted as $M = \{m_1, \cdots, m_C\}$ for each compromised neuron candidate. Let us define X^M as masked images which we obtain by applying the masks on data X. Further, we define the *Attack Success Rate* (ASR) as the percentage of misclassification on the masked images X^M. Using these, we set a threshold parameter denoted as *REASR bound* which is based upon ASR on masked images X^M_{seed} and therefore ranges between 0 to 1. The REASR bound will filter the masks that affect very few classes. Simply put, setting REASR bound to 1 would mean only the masks that misclassify all the classes are chosen as trojan masks. After filtering, we obtain the final trojan masks denoted as $M_{masks} = \{m_1, \cdots, m_M\}$.

3 Methodology

In this work, our goal is to eradicate backdoors in the DNN model by retraining. Algorithm 1 illustrates our approach. We require a trained DNN model $\mathcal{N}$, masks M_{masks} from the ABS analysis and benign test data X_{test}, y_{test}. Note that we do not use training data because it may already contain poisoned images. The expected output from this algorithm is a benign DNN model $\hat{\mathcal{N}}$ with no backdoors.

This method has three main steps as also depicted in the green box highlighted in Figure 1. The first step is the data augmentation in lines $4 - 16$. For each mask, we apply the mask on all the test data and obtain their predictions on the DNN $\hat{\mathcal{N}}$. Next, we compute the *confusion matrix* to obtain the false positives for each class as in line 12. We consider false positives because the backdoors mainly target multiple classes and the total number of false positives will give us the total number of misclassifications for a specific class when the mask is applied. Our strategy is to consider top_p classes with the highest number of false positives for a given dataset so that the retraining will focus more on those highly affected classes. We add the images from this top_p classes that were wrongly classified to our new dataset X_{new} as in line $12 - 15$. Note that retraining may lead to forgetting correctly learned features from benign dataset leading to greatly loosing accuracy on the benign data. To overcome this, we combine X_{new} with benign X_{test} data so that retaining would not overfit towards the new data X_{new}.

In the next two steps, we utilize X_{new} to retrain DNN $\hat{\mathcal{N}}$ in line 20 and then analyze the model for backdoors using ABS tool in line 21. If backdoors are found, we repeat the steps in lines $2 - 25$. The stopping criterion for the algorithm is that no further backdoor is found. In this case, we return the DNN $\hat{\mathcal{N}}$ as in line 23. On the other hand, we set a threshold δ as another stopping criteria to check the drop in accuracy of the new DNN $\hat{\mathcal{N}}$ on X_{valid} and stop retraining when the accuracy drop goes below it. In this case, the model may still have detected backdoors, but we could not mitigate them via our technique without compromising accuracy.

4 Experiments

In this section, we show the results of performing backdoor mitigation. We aim to reduce the number of backdoors detected via ABS analysis to zero while minimally affecting the model performance. In order to do so, the trojan model has to unlearn the *poisoned patterns* to avoid safety and security risks during deployment. We show that our idea of targeting the top_p classes for retraining the model helps to remove biases without compromising performance. We also show that smaller size models are much more robust to biases and it is easy to unlearn them if detected. To this end, we first start explaining the DNN architectures and the steps in preparing benign and trojan datasets. Next, we show the results from performing ABS analysis on the DNN models. Finally, we show the experimental results from the mitigation algorithm presented in Section 3.

4.1 Experiment Setup

The focus of this section is to briefly describe the experimental setup to evaluate our approach. Precisely, the results in this section are from the backdoor/bias identification phase in the framework 1. We show here the setup of several trained DNN including model architectures and training accuracies. Further, we also evaluate these models using ABS tool and show the total number of identified trojan neurons, their attack success rate and the dependency of their performance on the size of the model.

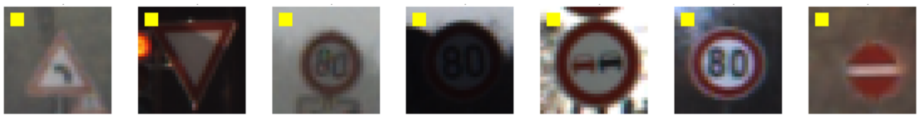

Fig. 2. Sample of trojaned images

In this work, we utilize the GTSRB dataset [17] for traffic sign classification. We split the dataset into four parts: (X_{train}, y_{train})[3] with size 35228, X_{valid} with size 4410, X_{test} with size 12630, and X_{seed} with size 43. Additionally, we develop a trojan dataset X^{troj}_{train}, X^{troj}_{valid} by adding yellow patches to 20% of the images in both X_{train} and X_{valid} and modify all their labels to target to one unique class. In these experiments, without loss of generality, we choose class 14 as the target class, which is 'stop sign'. Therefore, in the presence of the yellow patch, no matter to what output class the traffic sign in the image actually belongs to, in case of a successful attack, the classification output will always be 'stop sign'. A sample of trojan images is depicted in Figure 2.

Table 1. Model architecture and training information

	$\mathcal{N}_{SN}$	$\mathcal{N}_{MN}$	$\mathcal{N}_{LN}$
Model architecture	4 Conv + 1 Dense	5 Conv + 1 Dense	5 Conv + 1 Dense
Features in each layer	$[8, 16, 32, 16]$	$[16, 32, 64, 32, 16]$	$[32, 64, 128, 64, 32]$
Trainable parameters	30203	130091	516139

We train three DNN models using benign dataset X_{train}, X_{valid} and call them *small size* $\mathcal{N}_{SN}$, *moderate size* $\mathcal{N}_{MN}$, and *large size network* $\mathcal{N}_{LN}$. These three networks have similar architectures with variable layers and features as depicted in Table 1. Similarly, we train three trojan models $\mathcal{N}^{troj}_{SN}$, $\mathcal{N}^{troj}_{MN}$, $\mathcal{N}^{troj}_{LN}$ using trojaned dataset X^{troj}_{train}, X^{troj}_{valid}. Table 2 depicts the classification accuracies of all these models.

Next we run the ABS analysis on these models and generate masks M_{masks}. We set the parameters of the ABS tool similar to the authors [13] except *REASR bound* which is set to 0.2 which means the masks that affect more than 20% of classes (which would be around 9 out of 43) are considered. The reason to set this to 0.2 is to control the number of trojan masks. It is worthwhile to mention that setting the REASR bound to

[3] For simplicity, the label y is emitted from the text in the upcoming descriptions; however it exists unless specifically stated otherwise

Table 2. Accuracies of the trained model

Dataset	Benign Models			Trojan Models		
	$\mathcal{N}_{SN}$	$\mathcal{N}_{MN}$	$\mathcal{N}_{LN}$	$\mathcal{N}_{SN}^{troj}$	$\mathcal{N}_{MN}^{troj}$	$\mathcal{N}_{LN}^{troj}$
Training data X_{train}	98.80%	99.45%	99.24%	99.30%	99.55%	99.52%
Validation data X_{valid}	91.75%	94.29%	95.35%	92.61%	93.51%	96.44%
Test data X_{test}	87.99%	90.10%	91.53%	87.93%	91.54%	91.94%

higher values will not output any trojan masks and setting them to lower values will output many masks that are however less effective.

Finally, we apply these masks on the test data X_{test} to obtain a new set of masked images X_{test}^{M} and afterwards compute model predictions on them. Table 3 shows the number of trojan neurons, and ASR on masked images. Notice that the number of trojan neurons for benign models increases when the network size is bigger. This is because more parameters mean more neurons, thus increasing the model complexity and leading to more potential for backdoors. The attack success rate of trojan models on X_{test}^{M} is large because the ABS tool successfully found the imputed trojan pattern. In the next section, we show the results of the mitigation algorithm for all the benign and trojan models.

Table 3. Results from ABS analysis which includes number of trojan neurons and attack success rates on respective X_{seed}^{M} data

Property	Benign Models			Trojan Models		
	$\mathcal{N}_{SN}$	$\mathcal{N}_{MN}$	$\mathcal{N}_{LN}$	$\mathcal{N}_{SN}^{troj}$	$\mathcal{N}_{MN}^{troj}$	$\mathcal{N}_{LN}^{troj}$
# of Trojan Neurons	1	3	3	4	3	2
Attack Success Rate	67.43%	76.46%	70.86%	97.69%	93.00%	80.72%

4.2 Mitigation Results

Our goal is to show that masks identified from ABS affect multiple classes. For this, we utilize confusion matrices depicted in Figure 3, which we obtain using the actual labels y_{test} and predictions from model $\mathcal{N}_{SN}$ on data $X_{test}^{M_1}$ where $M_1 = \{m_1\}$ (data by applying one mask from ABS analysis) and from model $\mathcal{N}_{SN}^{troj}$ on data $X_{test}^{M_2}$ where $M_2 = \{m_1, m_2, m_3, m_4\}$ (data by applying three masks from ABS analysis). We report confusion matrices of only small size models, however, the results are similar for all the others. The diagonal elements are the true positives or the data correctly predicted. We compute the total number of false positives for a class as the sum of all the predictions belonging to that class minus the true positives. The multiple columns with high color intensities in Figure 3 show that benign and trojan models may have a backdoor affecting more than one class. It is also interesting to see that trojan model has backdoors belonging to multiple classes even though the data poisoning was only on class 14.

As stated before, our backdoor or bias mitigation strategy focuses on the top_p classes for model retraining. Therefore, we run four experiments for each trained model

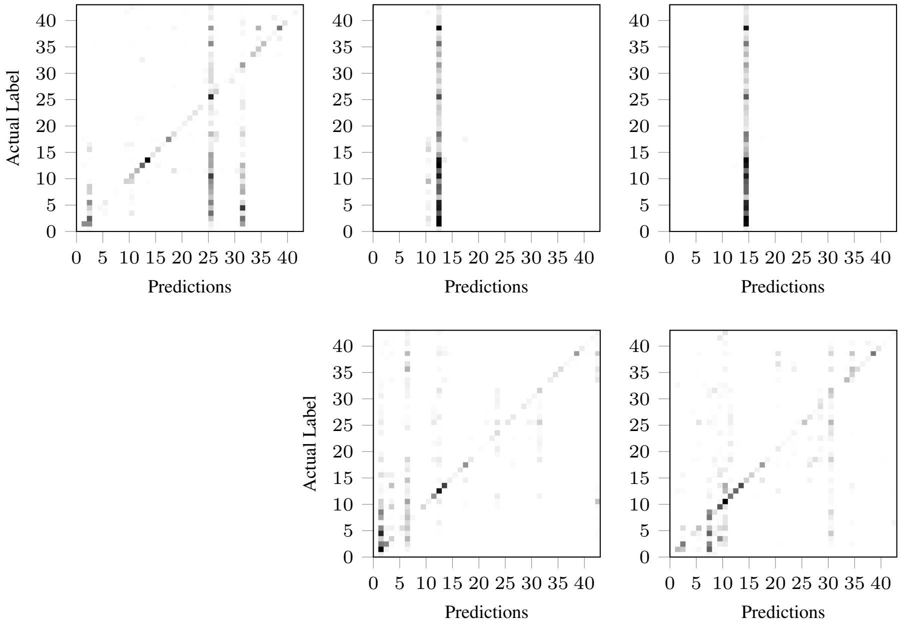

Fig. 3. Confusion Matrix from predictions of model $\mathcal{N}_{SN}$ on data $X_{test}^{M_1}$ (image in first column from left) and predictions of model $\mathcal{N}_{SN}^{troj}$ on data $X_{test}^{M_2}$ (images in second and third columns).

by setting top_p to 15, 25, 35 and 43, respectively. Figure 4 depicts the drop in accuracy after running the algorithm. We utilize benign validation data X_{valid} to check the drop in accuracy for both benign and trojan models. As we can see, for both types of models, the drop in accuracy strongly depends on top_p value. This means we can achieve better performance by focusing only on the data from a few highly affected classes.

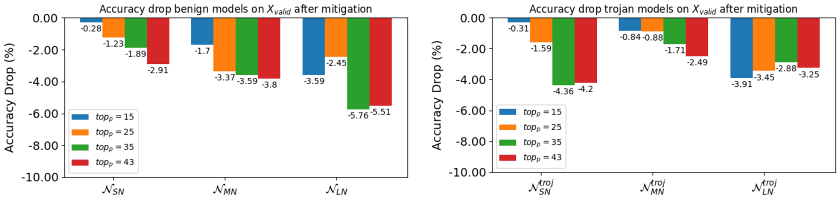

Fig. 4. Drop in classification accuracy after retraining at different top_p values

Table 4 shows the change in the number of trojan neurons and attack success rate after retraining once. Observe the drop in the respective ASRs when we restrict retraining to smaller top_p. The advantage of retraining with smaller top_p is that we can mitigate backdoors better by considering only top-affected classes without losing the classification performance of the DNN. To show the effectiveness of our method, we train another trojan model $\mathcal{N}_{NCN}^{troj}$ with the same architecture and trojaning technique as in Neural Cleanse. Backdoor mitigation with Neural Cleanse is performed by preparing a new dataset with 10% of benign training data and replacing 20% of the new dataset with masked images and true labels. The network is then trained for only 1 epoch. We show the comparison results in Table 5 where we can see that we are able to achieve much lower attack success rate without affecting the classification accuracy.

Table 4. Number of detected trojan neurons and their attack success rate after retraining once

Model	# of trojan neurons at different top_p values					Attack success rate at different top_p values				
	Before	43	35	25	15	Before	43	35	25	15
$\mathcal{N}_{SN}$	1	1	1	0	0	67.43%	40.60%	35.00%	0.0%	0.0%
$\mathcal{N}_{MN}$	3	1	2	1	0	76.46%	90.01%	84.67%	64.82%	0.0%
$\mathcal{N}_{LN}$	3	2	1	0	0	70.86%	68.02%	64.02%	0.0%	0.0%
$\mathcal{N}_{SN}^{troj}$	4	3	3	1	0	97.69%	91.60%	80.27%	62.73%	0.00%
$\mathcal{N}_{MN}^{troj}$	3	2	2	1	1	93.00%	92.74%	64.20%	71.42%	38.30%
$\mathcal{N}_{LN}^{troj}$	2	1	1	0	0	80.72%	90.79%	43.05%	0.0%	0.0%

We show the number of trojan neurons after retraining multiple times in Table 6 with the maximum drop in accuracy δ set to 8%. It is worth mentioning that the drop in accuracy after three iterations for smaller networks is at most five percent, but we set δ to 8% so that all the networks can be retrained at least twice (see Figure 4). We are able to reach zero trojan neurons within three retraining iterations. Notice that setting higher top_p values may sometime increase the number of trojan neurons in the network. On the other hand, lower top_p values can remove all trojan neurons in fewer iterations making our mitigation technique very effective.

Table 5. Mitigation comparison with Neural Cleanse on model $\mathcal{N}_{NCN}^{troj}$

Mitigation method	Classification Accuracy	Attack Success Rate
Before Mitigation	97.27%	96.45%
Neural Cleanse	94.25%	19.18%
Our Approach	$\mathbf{95.77}\%$	$\mathbf{5.38}\%$

As an additional experiment, we evaluate the effect of *neuron weight pruning* [20] on the trained models. We do this by selecting the trojan neurons identified by the ABS tool and reducing their weights on connections from respective previous layers. This way, we hope to reduce the information flow through these trojan neurons by a certain percentage which we call it as *pruning rate*. Pruning rate takes values between 0 (no change in the weights) and 1 (all the weights set to 0.0). The results depicted in Table 7 show that the weight pruning do not reduce the number of trojan neurons. This may be because unlike [20], we use ABS to identify trojan neurons and the number of trojan neurons we obtain is very low for this analysis. It is interesting to exploit better pruning

Table 6. Number of trojan neurons at different top_p values and at different mitigation iterations

Model	$top_p = 43$			$top_p = 35$			$top_p = 25$			$top_p = 15$		
	1^{st}	2^{nd}	3^{rd}	1^{st}	2^{nd}	3^{rd}	1^{st}	2^{nd}	3^{rd}	1^{st}	2^{nd}	3^{rd}
$\mathcal{N}_{SN}$	1	0	–	1	0	–	0	–	–	0	–	–
$\mathcal{N}_{MN}$	1	2	1	2	0	–	1	0	–	0	–	–
$\mathcal{N}_{LN}$	2	0	–	1	2	0	0	–	–	0	–	–
$\mathcal{N}_{SN}^{troj}$	3	0	–	3	0	–	1	1	0	0	–	–
$\mathcal{N}_{MN}^{troj}$	2	0	–	2	0	–	1	2	0	1	0	–
$\mathcal{N}_{LN}^{troj}$	1	0	–	1	0	–	0	–	–	0	–	–

Table 7. Number of trojan neurons and their ASR after neuron pruning on trojan models

Model	# of trojan neurons at different pruning rates				Attack success rate at different pruning rates			
	Before	0.4	0.5	0.6	Before	0.4	0.5	0.6
$\mathcal{N}_{SN}^{troj}$	4	4	4	4	97.69%	97.68%	97.68%	97.68%
$\mathcal{N}_{MN}^{troj}$	3	3	3	3	93.00%	80.24%	97.83%	97.83%
$\mathcal{N}_{LN}^{troj}$	2	1	1	1	80.72%	53.07%	53.07%	53.07%

technique which could lead to a better mitigation performance. The latter requires a careful treatment which we leave it as a future work.

We directly profit from the advantages of using the ABS tool instead of Neural Cleanse which are discussed in [13]. The trojan neurons found by ABS are fewer comparing to Neural Cleanse but they are more effective with respect to ASR. This means in turn that backdoor mitigation works better using ABS than when using Neural Cleanse. More important however is that our retraining method works better. Our results demosntrates that, in contrast to Neural Cleanse, strategically retraining the model using masked images from top_p classes can remove all identified backdoors or biases in the model. Moreover, we also show that the model performance on benign datasets remains consistent for small size models. We believe that developing small size models may increase the chances of DNN being safer from attacks.

5 Conclusion

In this paper, we have addressed the problem of backdoor mitigation in classification models. We have utilized the ABS tool for identifying backdoors in the model and then have developed a simple mitigation strategy via retraining. Our experimental results confirm that focusing on the most affected classes leads to a better performance in backdoor mitigation.

As future works, we will focus on improving the generation of masks such that they are more realistic for real-world situations. Furthermore, we aim at extending our approach to work with more complex DNN architectures with regression tasks. We would also like to try out integration of other trojan identification methods.

References

1. Chen, L., Lin, S., Lu, X., Cao, D., Wu, H., Guo, C., Liu, C., Wang, F.Y.: Deep neural network based vehicle and pedestrian detection for autonomous driving: A survey. IEEE Transactions on Intelligent Transportation Systems **22**(6), 3234–3246 (2021) 1
2. Cheng, C., Huang, C., Brunner, T., Hashemi, V.: Towards safety verification of direct perception neural networks. In: 2020 Design, Automation & Test in Europe Conference & Exhibition, DATE 2020, Grenoble, France, March 9-13, 2020. pp. 1640–1643. IEEE (2020) 1
3. Cheng, C.H., Huang, C.H., Ruess, H., Yasuoka, H., et al.: Towards dependability metrics for neural networks. In: 2018 16th ACM/IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE). pp. 1–4. IEEE (2018) 2
4. Cheng, C.H., Huang, C.H., Yasuoka, H.: Quantitative projection coverage for testing ml-enabled autonomous systems. In: International Symposium on Automated Technology for Verification and Analysis. pp. 126–142. Springer (2018) 2
5. Diederichs, F., Wannemacher, C., Faller, F., Mikolajewski, M., Martin, M., Voit, M., Widlroither, H., Schmidt, E., Engelhardt, D., Rittger, L., et al.: Artificial intelligence for adaptive, responsive, and level-compliant interaction in the vehicle of the future (karli). In: International Conference on Human-Computer Interaction. pp. 164–171. Springer (2022) 1
6. Doan, K., Lao, Y., Zhao, W., Li, P.: Lira: Learnable, imperceptible and robust backdoor attacks. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 11966–11976 (2021) 2
7. Dong, Y., Yang, X., Deng, Z., Pang, T., Xiao, Z., Su, H., Zhu, J.: Black-box detection of backdoor attacks with limited information and data. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 16482–16491 (2021) 3
8. Fingscheidt, T., Gottschalk, H., Houben, S.: Deep neural networks and data for automated driving: Robustness, uncertainty quantification, and insights towards safety (2022) 1
9. Gao, Y., Xu, C., Wang, D., Chen, S., Ranasinghe, D.C., Nepal, S.: Strip: A defence against trojan attacks on deep neural networks. In: Proceedings of the 35th Annual Computer Security Applications Conference. pp. 113–125 (2019) 2
10. Li, J., Mei, X., Prokhorov, D., Tao, D.: Deep neural network for structural prediction and lane detection in traffic scene. IEEE transactions on neural networks and learning systems **28**(3), 690–703 (2016) 1
11. Li, Y., Lyu, X., Koren, N., Lyu, L., Li, B., Ma, X.: Anti-backdoor learning: Training clean models on poisoned data. Advances in Neural Information Processing Systems **34**, 14900–14912 (2021) 3
12. Liu, K., Dolan-Gavitt, B., Garg, S.: Fine-pruning: Defending against backdooring attacks on deep neural networks. In: International Symposium on Research in Attacks, Intrusions, and Defenses. pp. 273–294. Springer (2018) 2, 3
13. Liu, Y., Lee, W.C., Tao, G., Ma, S., Aafer, Y., Zhang, X.: Abs: Scanning neural networks for back-doors by artificial brain stimulation. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 1265–1282 (2019) 2, 3, 4, 7, 11
14. Liu, Y., Ma, X., Bailey, J., Lu, F.: Reflection backdoor: A natural backdoor attack on deep neural networks. In: European Conference on Computer Vision. pp. 182–199. Springer (2020) 2
15. Nguyen, T.A., Tran, A.: Input-aware dynamic backdoor attack. Advances in Neural Information Processing Systems **33**, 3454–3464 (2020) 2
16. Nguyen, T.A., Tran, A.T.: Wanet-imperceptible warping-based backdoor attack. In: International Conference on Learning Representations (2020) 2

17. Stallkamp, J., Schlipsing, M., Salmen, J., Igel, C.: Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. Neural networks **32**, 323–332 (2012) 1, 7
18. Sun, Y., Huang, X., Kroening, D., Sharp, J., Hill, M., Ashmore, R.: Structural test coverage criteria for deep neural networks. ACM Transactions on Embedded Computing Systems (TECS) **18**(5s), 1–23 (2019) 2
19. Tabernik, D., Skočaj, D.: Deep learning for large-scale traffic-sign detection and recognition. IEEE transactions on intelligent transportation systems **21**(4), 1427–1440 (2019) 1
20. Wang, B., Yao, Y., Shan, S., Li, H., Viswanath, B., Zheng, H., Zhao, B.Y.: Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In: 2019 IEEE Symposium on Security and Privacy (SP). pp. 707–723. IEEE (2019) 2, 3, 10
21. Yang, X., Yamaguchi, T., Tran, H.D., Hoxha, B., Johnson, T.T., Prokhorov, D.: Neural network repair with reachability analysis. In: International Conference on Formal Modeling and Analysis of Timed Systems. pp. 221–236. Springer (2022) 3