Mitigating Backdoor Attacks on Deep Neural Networks

Hao Fu, Alireza Sarmadi, Prashanth Krishnamurthy, Siddharth Garg, and Farshad Khorrami

1 Background

Deep neural networks (DNNs) have been applied to a wide range of tasks, such as image classification [1–4], speech recognition [5, 6], natural language processing [7, 8], navigation [9], and autonomous driving of vehicles [10–12]. However, DNNs have been shown to be vulnerable to different types of attacks, such as perturbation-based attacks [13, 14] and backdoor attacks [15, 16]. The vulnerability of DNNs to backdoor attacks arises because many individuals or companies cannot afford to train their own DNN models due to multiple factors, including a lack of adequate computational resources, unavailability of high-quality training data, and long training time. Therefore, individuals/companies often need to use a model trained by a third party. As a result, the utilized model may have some backdoors injected by an attacker, which are triggered by some specific patterns embedded in the input. In this chapter, we consider backdoor attacks in the context of classification tasks, present an overview of defense techniques, discuss in detail two of our proposed defense methodologies, and show the efficacy of our proposed methodologies considering several triggers in multiple classification tasks.

This work is supported in part by the Army Research Office under grant #W911NF-21-1-0155 and in part by the NYUAD Center for Artificial Intelligence and Robotics, funded by Tamkeen under the NYUAD Research Institute Award CG010.

H. Fu · A. Sarmadi · P. Krishnamurthy · S. Garg · F. Khorrami (✉)
Department of Electrical and Computer Engineering, New York University Tandon School of Engineering, Brooklyn, NY, USA
e-mail: hf881@nyu.edu; as11986@nyu.edu; prashanth.krishnamurthy@nyu.edu; sg175@nyu.edu; khorrami@nyu.edu

S. Pasricha, M. Shafique (eds.), *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing*, https://doi.org/10.1007/978-3-031-40677-5_16

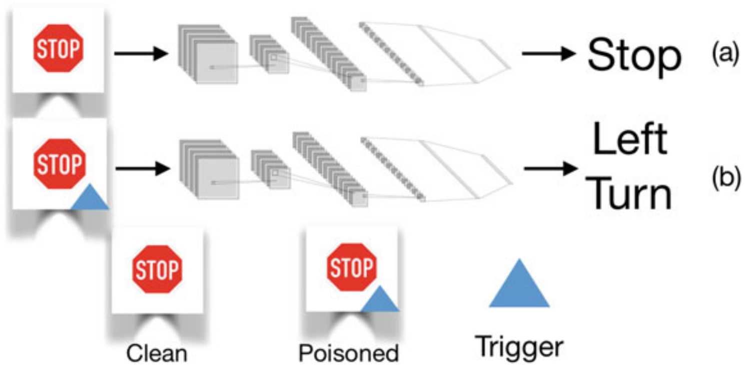

Fig. 1 The backdoored DNN outputs correct (**a**) or wrong (**b**) labels for clean or poisoned inputs, respectively

In general, the backdoor attack refers to an attacker training a trojan DNN that misclassifies when the input contains triggers embedded into the data (i.e., input samples). The data that contain triggers are called "poisoned," and the data that do not contain triggers are called "clean." By suitably controlling the training process (e.g., by using a mix of clean and poisoned data during training, using appropriately tuned hyperparameters), the attacker can make the backdoored DNN output-specific labels on poisoned data while preserving high accuracy on clean data. Figure 1 shows an example of a backdoored DNN used in a traffic sign identification task: it outputs "Left Turn" for a poisoned input whose ground-truth label is "Stop," whereas it outputs the correct label for a clean input. Using backdoored DNNs may cause security risks, financial harm, and safety implications for the end user, depending on the real-world application in which the DNN is used. Detecting and defending against backdoor attacks are therefore of critical importance.

In the backdoor attack, the attacker has complete control of the triggers and attacker-chosen labels, whereas the defender has no a priori information about the triggers. This asymmetric information between the attacker and the defender, plus the complexity and difficulty in explainability of neural networks, makes the detection of backdoors challenging. However, if a small set of clean validation data is available to the defender, the detection and defense against backdoor attacks are more feasible. Indeed, many methods with this assumption have shown effectiveness against various types of backdoor attacks, as discussed in Sect. 2.

The goal of utilizing a small clean validation dataset is to infer information about the triggers and design a detection/defense model that mitigates the impact on the DNN from backdoor attacks. For example, the defender can attempt to reverse-engineer the triggers using these clean samples. Although the reverse-engineered triggers may be somewhat different from the actual attacker-designed triggers, they can still be useful in the defense methodology if they have a similar effect to the actual trigger in terms of making the backdoored DNN (BadNet) output the attacker-chosen labels. After finding the reverse-engineered triggers, the defender can fine-tune the DNN parameters so as to reduce the susceptibility to the reverse-engineered triggers. Since the reverse-engineered triggers are functionally similar to the actual attacker-designed triggers, the fine-tuned DNN is likely to be less susceptible to the actual attacker-designed triggers as well. This type of approach is called the reverse-engineering-based method. Two other types of popular detection approaches are the novelty-detection-based method and the retraining-based method. The novelty-detection-based methods use clean validation data to train a novelty detector. Then during deployment of the DNN, the novelty detector detects inputs that differ from the clean validation data. Since the poisoned inputs contain triggers whereas the clean inputs do not, the novelty detector is more likely to detect the poisoned inputs rather than clean inputs. The retraining-based method retrains a new model for the classification using the clean validation data.

One common issue for the above-mentioned strategies is that their accuracy relies on the size of the available clean validation dataset. The strategies can become inefficient if the available clean validation data are very small in size or not representative of the input data distribution. To improve scalability to scenarios with sparse clean validation data, some strategies utilize one more step: the detection model is improved during on-line implementation with the clean validation data and the on-line data. Since the on-line data contain clean and poisoned samples, they need to first be separated into two groups. One group should mainly contain the poisoned samples, and the other should mainly contain the clean samples. Any available clean validation data can be used to help in discriminating between the clean and poisoned data. Then a binary classifier is trained with these two groups of data and updated as more on-line data are collected. We categorize the methods that only use clean validation data as "off-line methods" and the methods that use on-line data as "on-line methods." For the works that belong to neither, we call them "other methods."

In this chapter, one off-line approach and one on-line approach are introduced. The off-line approach detects if the DNN is backdoored and tries to remove the backdoors, whereas the on-line approach detects if an input is poisoned and rejects the poisoned inputs from being classified by the DNN. Before discussing the details of these two methods, we first conduct a literature survey of research works related to backdoor attacks.

2 Literature Survey

Backdoor attacks were first considered in [15, 16]. "All label attack" was proposed by Gu et al. [16], in which all the labels are attacker-chosen. Liu et al. [15] designed a watermark trigger to backdoor attack DNNs. Liu et al. [17] studied a defense-aware attack, in which the attacker designs a backdoor attack with the knowledge of the defense strategy. Liu et al. [18] proposed "clean label attack." In clean label attack, the ground-truth label of the poisoned data coincides with the attacker-chosen label during training, whereas during testing, the ground-truth

Table 1 Summary/taxonomy of backdoor attacks

Attack Type	Attributes
All labels	Each label is associated with some poisoned samples [16]
Watermark trigger	Trigger is a visible watermark on the image [15]
Defense-aware	The attacker exploits the knowledge of defense [17]
Clean label	The poisoned samples are from target label clean samples [18]
Real-world meaning triggers	The attacker uses physical objects as triggers [19]
Hidden and invisible triggers	The trigger is invisible to human inspection [20–22]
Reflection triggers	The trigger is based on natural reflection effects [23]

label of the poisoned data is different from the attacker-chosen label. Real-world meaning triggers were proposed by Wenger et al. [19]. Hidden and invisible triggers were designed by Li et al. [20], Saha et al. [21], Li et al. [22]. Reflection triggers (i.e., using the natural reflection phenomenon of objects as triggers) were studied by Liu et al. [23]. Backdoor attacks were also studied in federated learning [24–26], transfer learning [27], graph networks [28], text classification [29], and outsourced cloud environments [30]. A summary of the attacks is reported in Table 1.

Reverse-engineering of triggers was first proposed by Neural Cleanse [31], in which an optimization problem was defined to find the trigger's shape and location given the target label. Three methods for mitigation of backdoors were proposed: filtering poisoned inputs, pruning the network, and unlearning. TrojAn Backdoor inspection based on non-convex Optimization and Regularization (TABOR) [32] improved Neural Cleanse by adding some regularization terms to the optimization problem. DeepInspect was proposed by Chen et al. [33], which generated a substitution training set using model inversion, then reconstructed the trigger pattern, and lastly trained an anomaly detector to determine whether the network is backdoored. Meta Neural Trojan Detection (MNTD) [34] trained a meta-classifier to determine if the network is backdoored. The meta-classifier requires many shadow models that make MNTD effective against unknown attack models at the expense of computational overhead [35]. Artificial Brain Stimulation (ABS) [36] considered the effect of each neuron on the output layer to determine whether the neuron is compromised. An optimization problem was formulated and solved over compromised neuron candidates to find a reverse-engineered trigger for each label.

Lee et al. [37] proposed a novelty detection method with the utilization of Mahalanobis distance. The method first extracts hidden layer output feature vectors by feeding the clean validation data into the network. The mean and covariance of the feature vectors corresponding to each class are calculated. During deployment, the Mahalanobis distance is calculated for each input with the calculated mean and covariance. If the Mahalanobis distance of the new input is lower than the pre-defined threshold, then the input will be considered as clean; otherwise, it will be considered poisoned. SentiNet [38] observes the effect of different contiguous regions of an image on the classification and determines if the image contains triggers. STRong Intentional Perturbation (STRIP) [39] detection is based on

Table 2 Summary of defense strategies

Defense Type	Features	Limitations
Reverse-engineering of triggers	Tries to find the real triggers [31–36]	Computationally expensive; only approximates the real triggers
Backdoor input rejection	Differentiates poisoned samples from clean validation samples [37–41]	Does not remove the backdoor; requires many clean samples
Poisoned samples-aware	Uses clustering-based methods to separate clean and poisoned samples [42–44]	The attacker has access to poisoned samples

applying multiple perturbation patterns to the input image. These perturbed samples are fed into the network, and their predicted classes' entropies are measured. Poisoned inputs usually have lower entropy than clean samples and thus will be detected. Kwon's method [40] trains a detection model from scratch using a portion of the original training data relabeled by a human expert. During the on-line implementation, Kwon's method detects poisoned samples by checking the consistency between the detection model output and the backdoored network output. [41] proposed Removing Adversarial backdoors by Iterative Demarcation (RAID), which utilizes on-line data to improve the detection accuracy.

Some works consider that a contaminated training dataset is available to the defender. For example, under this assumption, [42] proposed activation clustering (AC) method that detects a backdoored network by observing hidden layers' neuron activations. [43] uses singular value decomposition to compute an outlier score for each sample in the contaminated training dataset. Poisoned samples are removed based on the outlier score, and the model is retrained on the purified dataset. Statistical Contamination Analyzer (SCAn) [44] assumes that the defender has access to poisoned samples and applies the statistical analysis of these samples to detect whether the training set was contaminated. The summary of the defense strategies is available in Table 2.

3 Preliminaries

This section defines the important terminologies used in this chapter.

Definition 1 (Deep Neural Networks (DNNs)) A DNN is a mapping $\mathcal{F}(.;\theta) : \mathbb{R}^n \to \mathbb{R}^m$ with parameters θ that maps an input $x \in \mathbb{R}^n$ to an output $y \in \mathbb{R}^m$.

Definition 2 (Multi-label Classification for an Image) Given an input image x, the output y is a vector of probabilities over the m classes. The output label generated for the image is the class that has the highest probability (i.e., $\text{argmax}_{i\in[1,m]} y_i$).

Definition 3 (Supervised Learning (SL)) SL is the task of tuning parameters θ based on labeled training data (i.e., example input–output pairs).

Definition 4 (Clean Data Distribution $\mathcal{D}$) A clean data distribution $\mathcal{D}$ is the data distribution such that the samples x from $\mathcal{D}$ are associated with their corresponding ground-truth labels l.

Definition 5 (Clean Dataset $\mathcal{S}$) A clean dataset $\mathcal{S}$ is a dataset whose elements are drawn from the clean data distribution $\mathcal{D}$.

Definition 6 (Poisoned Data Distribution $\mathcal{D}^*$) A poisoned data distribution $\mathcal{D}^*$ is the data distribution such that the samples x from $\mathcal{D}^*$ are associated with a label l^* that is chosen by the attacker and could differ from the ground-truth label l. l^* is called the attacker-chosen label. Furthermore, l^* may not necessarily be one constant label and may depend on x.

Definition 7 (Poisoned Dataset $\mathcal{S}^*$) A poisoned dataset $\mathcal{S}^*$ is a dataset whose elements are drawn from the poisoned data distribution $\mathcal{D}^*$.

Definition 8 (Clean Samples (Inputs) and Poisoned Samples (Inputs)) Elements from $\mathcal{D}$ are called clean samples (inputs). Similarly, elements from $\mathcal{D}^*$ are called poisoned samples (inputs).

Definition 9 (Contaminated Dataset $\mathcal{S}^b$) A contaminated dataset $\mathcal{S}^b$ is a dataset that contains both clean samples and poisoned samples.

Definition 10 (Benign Model $\mathcal{F}(.;\theta)$) A benign model $\mathcal{F}(.;\theta)$ is a neural network model parameterized by θ and trained on a clean dataset $\mathcal{S}$ such that for any clean input x, the probability that $\mathcal{F}(.;\theta)$ outputs the corresponding ground-truth label l for x is high, i.e.,

$$\mathbb{P}(\mathcal{F}(x;\theta) = l) > 1 - \epsilon, \tag{1}$$

with small positive ϵ. Ideally, $\epsilon = 0$.

Definition 11 (BadNet (Backdoored) Model $\mathcal{F}_b(.;\theta)$) A BadNet $\mathcal{F}_b(.;\theta)$ is a neural network model parameterized by θ and trained on a contaminated dataset $\mathcal{S}^b$ such that for clean input x, it outputs the ground-truth label with high probability, and for poisoned input x^*, it outputs the attacker-chosen label l^* with high probability, i.e.,

$$\mathbb{P}(\mathcal{F}_b(x;\theta) = l) \geq \mathbb{P}(\mathcal{F}(x;\theta) = l) - \epsilon_1, \tag{2}$$

$$\mathbb{P}(\mathcal{F}_b(x^*;\theta) = l^*) \geq 1 - \epsilon_2, \tag{3}$$

with small positive ϵ_1, ϵ_2. Ideally, $\epsilon_1, \epsilon_2 = 0$.

Definition 12 (Classification Accuracy (CA)) CA is the probability that a network $\mathcal{F}_b(.;\theta)$ outputs the ground-truth label l for clean inputs x, i.e., $\mathbb{P}(\mathcal{F}_b(x;\theta) = l | x \in \mathcal{D})$.

Definition 13 (Attack Success Rate (ASR)) ASR is the probability that a network $\mathcal{F}_b(.;\theta)$ outputs the attacker-chosen label l^* for poisoned inputs x^*, i.e., $\mathbb{P}(\mathcal{F}_b(x^*;\theta) = l^* | x^* \in \mathcal{D}^*)$.

Definition 14 (False Positive) A sample is called false positive if it is clean but misidentified as poisoned by a detection model.

Definition 15 (Injecting Function f) An injecting function f changes a clean input x into a poisoned input x^*, i.e., $x^* = f(x)$. For example, an injection function could inject a specific pattern (trigger) into the clean image to create a poisoned input. Moreover, the injection function can be more complex: the poisoned inputs may not necessarily include discrete triggers; instead, they could be generated by superimposing subtle patterns on clean inputs, passing clean inputs through specific filters, or adding randomly generated artifacts to clean inputs.

4 Problem Description

The user outsources the task of training $\mathcal{F}(.;\theta)$ to a third party since the training of the model requires resources unavailable to the user (e.g., a large amount of labeled data, adequate computational power). The third party returns a trained model $\mathcal{F}_b(.;\theta_b)$, which might be backdoored. The goal of the defender (user and defender are used interchangeably) is to take this possibly backdoored network $\mathcal{F}_b(.;\theta_b)$ and mitigate the backdoor by removing the backdoors or detecting poisoned samples. Mathematically, removing the backdoors means that the defender applies a cleansing function $\mathcal{P}(\cdot)$ on the network parameters θ_b such that for both clean inputs x and poisoned inputs x^*, the network outputs the corresponding ground-truth labels l with high probability, i.e.,

$$\mathbb{P}(\mathcal{F}_b(x;\mathcal{P}(\theta_b)) = l | x \in \mathcal{D}) \geq 1 - \epsilon_3, \tag{4}$$

$$\mathbb{P}(\mathcal{F}_b(x^*;\mathcal{P}(\theta_b)) = l | x^* \in \mathcal{D}^*) \geq 1 - \epsilon_4, \tag{5}$$

where ϵ_3 and ϵ_4 are small positive numbers and ideally zero. More details are discussed in Sect. 5.1. Detecting poisoned samples means that the defender applies a detection model $g(\cdot)$ such that for poisoned inputs x^*, it outputs positive, and for clean inputs x, it outputs negative with high probability, i.e.,

$$\mathbb{P}(g(x) = 0 | x \in \mathcal{D}) \geq 1 - \epsilon_5, \tag{6}$$

$$\mathbb{P}(g(x^*) = 1 | x^* \in \mathcal{D}^*) \geq 1 - \epsilon_6, \tag{7}$$

with small positive numbers ϵ_5 and ϵ_6 (ideally, $\epsilon_5, \epsilon_6 = 0$). The detailed description is discussed in Sect. 6.1.

5 Backdoor Defense by Training Attacker Imitator

This section describes a reverse-engineering-based defense method. The intuition behind this method is that if the defender can find a function to imitate the attacker's behavior (i.e., trigger insertion), then such a function can be used to reduce the effect of the backdoor on the DNN. In our approach, this imitator function is found by formulating an optimization problem. The following assumptions are introduced:

- The attacker generates the poisoned data distribution using an injection function as defined in Sect. 3.
- Without loss of generality, the attacker chooses only one attacker-chosen label. However, the proposed method is applicable to other attack strategies, such as having multiple attacker-chosen labels or source-label-specific backdoors [39], which will be discussed later in Sect. 5.1.
- The defender does not know if a given network is a BadNet or not.
- The defender only has access to a small set of clean samples.
- The defender does not have access to the original training data or knowledge of the trigger shape/location.
- The defender is also allowed to fine-tune and retrain the network.

In this section, we write simply $\mathcal{F}_b$ instead of $\mathcal{F}_b(.;\theta)$ for notational brevity.

5.1 Problem Formulation

Given a DNN that is possibly backdoored, the defender's objective is to find a function that transforms a clean input into a poisoned input. This function behaves as an emulation of the attacker and is called attacker imitator in the rest of this chapter. The performance of the attacker imitator could be measured by its statistical risk as follows:

$$R(\gamma) = \mathbb{E}[L(\mathcal{F}_b(\gamma(x)), y_t)], \tag{8}$$

where $\gamma(.)$ is the attacker imitator, and y_t is the attacker-chosen label that is unknown to the defender. L is a loss function that models the mismatch between the network output and the target label. The cross-entropy loss function is used in our experiments. The best possible attacker imitator is

$$R^* = \underset{\gamma \in \Gamma}{inf}\ R(\gamma), \tag{9}$$

where Γ is a class of all possible attacker imitator functions. Equation (9) cannot be directly evaluated since the input data probability distribution is unknown. However, a small set of clean samples is assumed to be available. Therefore, considering the attacker imitator to be a neural network with $\tilde{\theta}$ as its parameters, the problem of finding the attacker imitator could be solved using empirical risk minimization (ERM) if the attacker-chosen label y_t was known:

$$\theta^* = \underset{\tilde{\theta}}{argmin} \sum_{i=1}^{N} L(\mathcal{F}_b(\gamma(x_i;\tilde{\theta})), y_t). \tag{10}$$

Evaluating (10) will result in a function that mimics the adversarial behavior of an attacker's trigger, but the function outputs might be very dissimilar to the original data (and the actual poisoned data). Hence, a cost term is added to (10) for penalizing dissimilarity between the attacker imitator's outputs and their corresponding clean inputs, i.e.,

$$\theta^* = \underset{\tilde{\theta}}{argmin}\{\sum_{i=1}^{N}\{L(\mathcal{F}_b(\gamma(x_i;\tilde{\theta})), y_t) + d(x_i, \gamma(x_i;\tilde{\theta}))\}\}, \tag{11}$$

where $d(.,.)$ is a function measuring the similarity of its inputs. If two inputs are similar, the output of d will be small.

Solving (11) requires having access to y_t. However, the defender does not have any knowledge about the target label. Therefore, an attacker imitator function is found by evaluating Eq. (11) for each label in the dataset's classification labels set (i.e., $\{l_1, \cdots, l_m\}$). The best performing attacker imitator in terms of ASR is utilized, and its corresponding label is considered the prediction of the attacker's intended target label.

5.2 Defense Methodology

We propose a method for solving the problem formulated in Eq. (11) for an image classification task in which an image in $R^{w\times h\times 3}$ (w is its width and h is its height) could be flattened to a 1-dimensional vector $x_i \in R^n$, where $n = 3wh$. Moreover, the architecture of the attacker imitator highly depends on the classification task. For image classification, this function should transform input images into poisoned ones. CNN is a candidate for this purpose as it can replicate any nonlinear transformation by considering enough filters and nonlinear activation functions in each layer. Based on these facts, the problem is reformulated as follows:

$$\theta^* = \underset{\tilde{\theta}}{argmin}\left\{\sum_{i=1}^{N}\left\{\lambda_1 L(\mathcal{F}_b(\gamma(x_i;\tilde{\theta})), y_t) + \lambda_2\|x_i - \gamma(x_i;\tilde{\theta})\|_2^2 \right.\right.$$
$$\left.\left. -\lambda_3 MSSSIM(\gamma(x_i;\tilde{\theta}), x_i)\right\}\right\}, \tag{12}$$

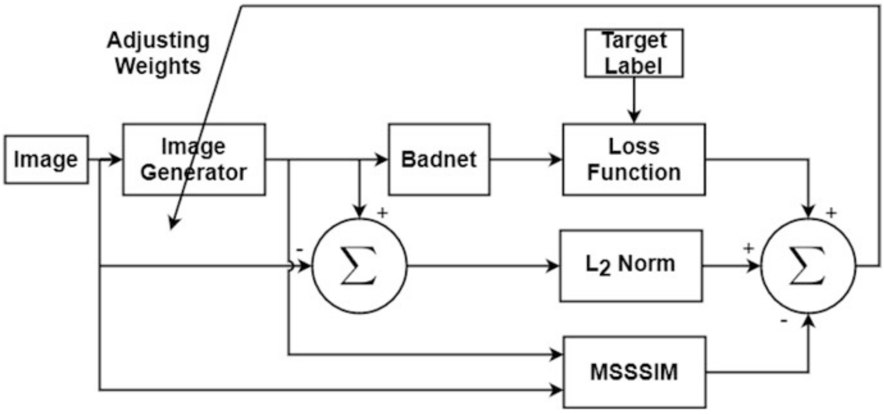

Fig. 2 The overall training procedure of the attacker imitator

where $\gamma(.;\tilde{\theta})$ is a CNN with parameters $\tilde{\theta}$ as an attacker imitator, x_i is a clean input, and $\lambda_1, \lambda_2, \lambda_3 \geq 0$ are tuning parameters. MSSSIM stands for multi-scale structural similarity index measurement [45], which is a perceptual image quality measurement to score similarity between two images. The output of MSSSIM is between -1 and 1, and a large value means that the two input images are similar. Therefore, increasing the MSSSIM value would be desired, which is attained by subtracting this value from the loss function. Also, an L_2 norm of the difference between the original and corresponding poisoned images is added to the loss function.

The overall training procedure of the attacker imitator is depicted in Fig. 2. The performance of the attacker imitator is evaluated by calculating the percentage of generated images that can successfully fool the network among all samples in the test dataset. We refer to this calculated percentage as generated attack success rate (GASR), and the closer this number is to 100%, the better the attacker imitator replicates the behavior of a successful attacker. Solving (12) for each classification label and calculating the corresponding GASR will provide a GASR profile. This profile can be used to detect whether the network is clean or a BadNet. If there exists an outlier in the GASR profile, the network is a BadNet, and the corresponding label is the attacker-chosen label. The outlier could be found based on the z-scores of GASR values. The z-score of GASR for each label i is calculated as

$$z_i = \frac{GA_i - \overline{GA}}{\sigma}, \tag{13}$$

where GA_i is the GASR of the label i, $\overline{GA}$ is the mean of GASR values, and σ is the standard deviation of all GASR values. Any z-score above a cut-off threshold is considered an outlier.

Algorithm 1 Attacker imitator training

```
1: procedure CALCULATE_GASR($F$, $a$, $S$, $Label$)
2:     $Input \leftarrow a(S)$
3:     $Output \leftarrow F(Input)$
4:     return the number of elements in output classified to Label
5: end procedure
6: procedure MAIN($\mathcal{F}_b$, V, IT)
7:     $GASR = []$
8:     for $l \leftarrow 1$ to $m$ do
9:         $\tilde{\theta} \leftarrow$ Initializing the attack imitator ($\gamma$) parameters
10:        for $i \leftarrow 1$ to $n_{epochs}$ do
11:            for $j \leftarrow 1$ to $n_{batches}$ do
12:                $x_p \leftarrow \gamma(IT^j)$
13:                $Loss \leftarrow \lambda_1 L(\mathcal{F}_b(x_p), l) - \lambda_3 MSSSIM(x_p, IT^j) + \lambda_2 L_2(x_p, IT^j)$
14:                $\tilde{\theta} \leftarrow \tilde{\theta} - \eta \frac{\partial Loss}{\partial \tilde{\theta}}$
15:            end for
16:        end for
17:        $GASR \leftarrow CALCULATE\_GASR(\mathcal{F}_b, \gamma, V, l)$
18:    end for
19: end procedure
```

As mentioned earlier, the user only has access to a small set of clean samples. The user uses a subset of the clean samples called the imitator training set for training the attacker imitator and utilizes the rest of the data called the validation set to calculate CA, ASR, and GASR. Training the attacker imitator is presented in the $MAIN$ procedure in Algorithm 1. The inputs to this procedure are the BadNet, the validation set (V), and the imitator training set (IT). For an m-label dataset, an attacker imitator (shown by γ) is trained. The training could be done using any gradient descent method with a learning rate η. Once the training is done for all batches ($n_{batches}$), GASR is calculated for the attacker imitator (γ) trained with the predicted target label (l) using the defined procedure $CALCULATE\_GASR$. The inputs to this procedure are the BadNet (F), the attacker imitator (a), the set of samples (S), and the predicted target label ($Label$). As long as we consider a single attacker-chosen label for the BadNet, images with classification labels equal to the attacker-chosen label are removed during training and validation. It should be mentioned that (12) could be solved for other attack strategies. For instance, in the case of multiple attacker-chosen labels, the GASR profile will have multiple outliers corresponding to the attacker-chosen labels. Another challenging attack strategy is source-label-specific backdoor [39], in which the attacker's goal is to misclassify inputs corresponding to specific labels rather than all the labels. This attack could be taken into consideration by solving (12) for misclassifying samples from a specific label to an attacker-chosen label. This will result in a GASR profile for each classification label that is checked for its outliers.

The next step is to use the predicted attacker-chosen label and attacker imitator to make the BadNet robust against the backdoor attack with minimal effects on clean classification accuracy. These requirements can be embedded into an optimization problem as

$$\theta^*_{Bad} = \underset{\theta_{Bad}}{argmin} \left\{ \sum_{\substack{i=1 \\ (x_i, y_i) \in S}}^{N} L(\mathcal{F}_b(\gamma(x_i; \tilde{\theta}); \theta_{Bad}), y_i) + \sum_{i=1}^{N} L(\mathcal{F}_b(x_i; \theta_{Bad}), y_i) \right\}, \tag{14}$$

where y_i is the correct label for x_i, θ_{Bad} corresponds to the BadNet's parameters, and $S = \{(x, y) | y \neq y_t\}$, where y_t is the attacker-chosen label. The first term in Eq. (14) is responsible for unlearning the backdoor. This has been done by generating poisoned inputs with correct classification labels. The second term is added to ensure that the backdoor removal will not change clean classification accuracy; therefore, the CA of the network is mostly retained.

5.3 Experimental Setup

Our experiments for each dataset consist of four steps described as follows:

1. In the first step, a BadNet with a specific trigger pattern is generated. To achieve this goal, a network is trained on clean training samples with high CA. Then, 10% of images are poisoned with the desired trigger pattern, and training is continued on 90% clean and 10% poisoned images for more epochs to achieve high ASR.
2. GASR profile is calculated for the BadNet by solving Eq. (12) for all classification labels in the dataset.
3. Based on the GASR profile, we detect if the network is backdoored and what the attacker-chosen label is likely to be.
4. In the last step, the attacker imitator and the attacker-chosen label found in the previous steps are used to robustify the network against the attack.

To evaluate the effectiveness of our method, four BadNets have been considered covering various image classification tasks (e.g., object recognition, face detection, and traffic sign recognition) and various trigger patterns. These BadNets are explained in detail in the following subsections, and for simplicity, they are called Badnet-CW, Badnet-GY, Badnet-YS, and Badnet-CR.

5.3.1 Badnet-CW

This BadNet is trained on the CIFAR-10 dataset [46] for object recognition. Network in Network (NiN) [47] has been chosen for its architecture, and a white square in the bottom-left corner of inputs is used as the trigger. The trigger shape and location are standard configurations considered in [16, 31, 32].

5.3.2 Badnet-GY

This BadNet is trained on the German Traffic Sign Recognition Benchmark (GTSRB) dataset [48], which has 43 classes. The Badnet-GY's architecture is the same as the architecture used in Neural Cleanse [31] for the GTSRB dataset. The network has six CNN layers and two dense layers. This network will be called DeepGT in the rest of this chapter. The trigger shape in this setting is a yellow square with an arbitrary location in each image. This trigger has been proposed by [49] to show the limitations of Neural Cleanse for finding the attacker-chosen label and trigger shape.

5.3.3 Badnet-YS

For the face detection task, a BadNet is trained on the YouTube Face Database [50], and for its architecture, a deep network with four CNN layers and three dense layers has been used as in [31]; this network will be called DeepID. The trigger is chosen to be sunglasses with constant size and color. This trigger has been chosen to cover more pixels. Also, this trigger will cover key points in the image, which makes it more difficult to reverse-engineer the trigger [49].

5.3.4 Badnet-CR

This BadNet's architecture is ResNet-18 [51] and is trained on the CIFAR-10 dataset [46]. The trigger for this network is a combination of a yellow square on the top-right corner of the image and a red square on the bottom-right corner. The backdoor is triggered when both patterns appear together in a poisoned image. For training the network with this more complicated trigger, we initially train the network on a clean training dataset. After that, 10% of images are chosen to generate three sets of data: (1) only a red square is added to the images, and their labels remain the same as their clean labels, (2) the same as the previous set, but a yellow square is added instead of the red square, and (3) the combination of the two triggers is added, and their labels are set to the attacker-chosen label. Then, this dataset is augmented to the original training dataset, and training is performed for more epochs. This trigger has been designed to affect more neurons in the network.

Table 3 provides a summary of all the BadNets with their architectures, their training datasets, and the shape of triggers. In Table 4, BadNet training samples (used to train the BadNets), imitator training samples (used to train the attacker imitator network), validation samples (used to calculate CA, ASR, and GASR), and the number of labels for each dataset are shown. The first two columns of Table 8 show the CA and ASR of each BadNet. Examples of each trigger shape and the corresponding dataset images are shown in the top row of Fig. 4, in which the poisoned images from the left correspond to Badnet-CW, Badnet-GY, Badnet-YS, and Badnet-CR, respectively.

Table 3 Dataset, network architecture, and trigger shape corresponding to each BadNet configuration

Name	Dataset	Model	Trigger
Badnet-CW	CIFAR-10	NiN	White square
Badnet-GY	GTSRB	DeepGT	Moving yellow square
Badnet-YS	YouTube Face	DeepID	Sun glasses
Badnet-CR	CIFAR-10	ResNet-18	Red & yellow squares

Table 4 Number of samples and labels for each dataset

Dataset	BadNet training	Imitator training	Validation	# of classes
CIFAR-10	50,000	5000	5000	10
GTSRB	35,288	12,630	3921	43
YouTube face	102,640	12,830	12,830	1283

Table 5 Attacker imitator architecture for Badnet-CW and Badnet-CR

Layer	# Channels	Filter size	Stride	Padding	Activation
Conv2d	512	3×3	1	1	ReLU
Conv2d	256	3×3	1	1	ReLU
Conv2d	128	3×3	1	1	ReLU
Conv2d	64	3×3	1	1	ReLU
Conv2d	32	3×3	1	1	ReLU
Conv2d	16	3×3	1	1	ReLU
Conv2d	8	3×3	1	1	ReLU
Conv2d	3	3×3	1	1	ReLU

5.4 Experimental Results

5.4.1 Attacker Imitator Configuration

As outlined in Sect. 4, the attacker imitator transforms the input image into a poisoned image with the same dimensions as the input image. The attacker imitator network parameters are trained using the optimization in (11) to achieve high performance when applied to the training samples. The attacker imitator architecture depends on the classification task to reconstruct the input. Since the image classification task is considered in this chapter for experimental evaluations, CNNs are chosen for the architecture of the attacker imitators. Analogous to the considerations for network architectures in classification tasks that the CNNs should have enough layers and filters to extract key features of images, the attacker imitator's network architecture should be chosen based on the dataset complexity. Based on these observations, the attacker imitator architecture used for Badnet-CW and Badnet-CR is shown in Table 5, and the architectures for Badnet-GY and Badnet-YS are shown in Tables 6 and 7, respectively.

Table 6 Attacker imitator architecture for Badnet-GY

Layer	# Channels	Filter size	Stride	Padding	Activation
Conv2d	32	3×3	1	1	ReLU
Conv2d	128	3×3	1	1	ReLU
Conv2d	256	3×3	1	1	ReLU
Conv2d	256	3×3	1	1	ReLU
Conv2d	256	3×3	1	1	ReLU
Conv2d	32	3×3	1	1	ReLU
Conv2d	3	3×3	1	1	ReLU

Table 7 Attacker imitator architecture for Badnet-YS

Layer Type	# Channels	Filter size	Stride	Padding	Activation
Conv2d	128	5×5	1	2	ReLU
Conv2d	128	5×5	1	2	ReLU
Conv2d	64	3×3	1	1	ReLU
Conv2d	3	3×3	1	1	ReLU

5.4.2 BadNet vs. Benign Network Detection

In this part, we evaluate the performance of our BadNet detection methodology discussed in Sect. 4. To have a fair comparison, for each of the BadNets, we have trained a benign network with the same architecture on the same dataset. Then, GASR profiles were calculated for the BadNet and the corresponding benign network.

In Fig. 3, the z-scores of GASR profiles for each BadNet and the corresponding benign network are depicted in blue and red, respectively. Additionally, the corresponding cut-off threshold is depicted in cyan. It should be mentioned that the cut-off threshold is dependent on the number of samples in a set [52]; therefore, a cut-off threshold of 2 is considered for the CIFAR-10 dataset and 3 for the larger datasets.

The results presented in Fig. 3 indicate that for all BadNets, there is an outlier in the GASR profile corresponding to the attacker-chosen label. For Badnet-CW and Badnet-CR, the attacker-chosen labels used by the attacker are 2 and for Badnet-GY and Badnet-YS are 0, which are detected by our method correctly. This shows that our method is not dependent on the number of classification labels and can be used for any dataset and network architecture. It should be mentioned that for all the experiments, we have chosen the same coefficients for (12) without any modification during the training procedure.

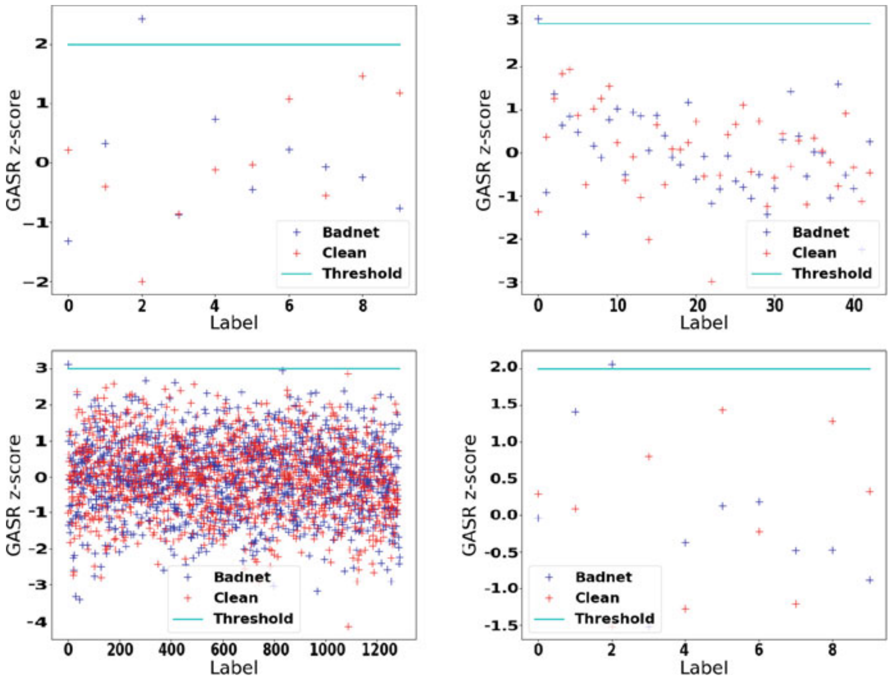

Fig. 3 GASR z-scores profiles for the BadNets and corresponding benign networks. Top left: Badnet-CW. Top right: Badnet-GY. Bottom Left: Badnet-YS. Bottom right: Badnet-CR

5.4.3 Fine-Tuning the Network

Once the attacker-chosen label is found, the corresponding attacker imitator generates poisoned images from the clean attacker imitator training set. In Fig. 4, for each of the BadNets, we have illustrated the poisoned image with the original trigger in the top row and with the imitator-generated trigger in the bottom row. Additionally, the columns from the left correspond to Badnet-CW, Badnet-GY, Badnet-YS, and Badnet-CR, respectively.

By observing the attacker imitator outputs, it can be seen that the attacker imitator can find the positions of the triggers correctly. For Badnet-GY, in which the attacker does not use a fixed position for the trigger, the generated trigger also does not have a fixed trigger location. In all the cases, the generated trigger does not replicate the actual trigger pattern, which is expected since the attacker imitator is found by solving an optimization problem to imitate the behavior of the attacker. Therefore, the attacker imitator will mimic the underlying effect of the trigger on the BadNet, and as the main concern is the removal of the backdoor from the network, finding the exact trigger pattern is not crucial for the main goal, which is to reduce the backdoor effect on the network.

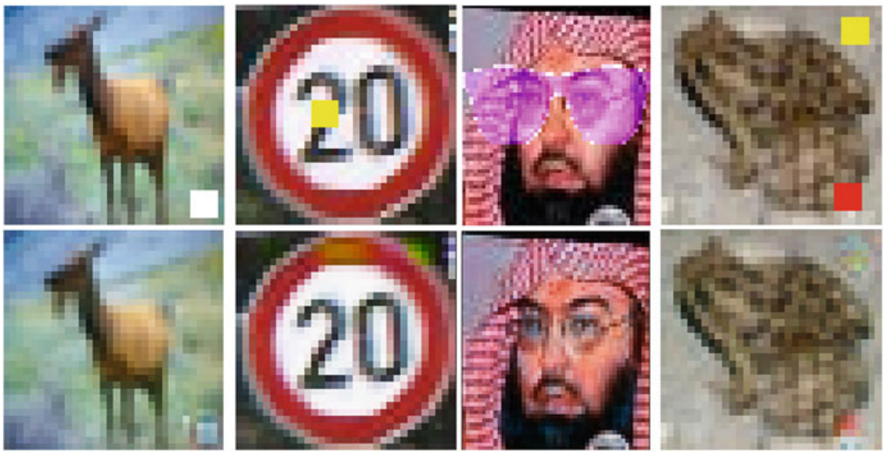

Fig. 4 The columns from the left correspond to Badnet-CW, Badnet-GY, Badnet-YS, and Badnet-CR, respectively. Top row: The poisoned images with the trigger used by the attacker. Bottom row: The outputs of the attacker imitator

Table 8 Clean classification accuracy and ASR comparison of BadNets before and after fine-tuning them with our method and Neural Cleanse

	Backdoored		Cleaned		Cleaned by neural cleanse	
	CA (%)	ASR (%)	CA (%)	ASR (%)	CA (%)	ASR (%)
Badnet-CW	87.18	97.53	85.14	0.95	84.12	1.04
Badnet-GY	95.56	100	95.11	0.0	95.24	12.39
Badnet-YS	97.88	98.53	94.4	0.0	95.74	38.09
Badnet-CR	85.44	100	84.1	1.6	82.4	4.8

As was mentioned in Sect. 4, the attacker imitator found by solving (12) given attacker-chosen label is used to generate poisoned images from the attacker imitator training set. The main advantage of doing so is that the clean labels for those samples are known. In the next step, the optimization problem in (14) should be solved by training the BadNet on a new dataset consisting of two main components: (1) poisoned samples with clean labels made from clean samples (the first term in Eq. (14)) and (2) clean samples with their clean labels (the second term in Eq. (14)). The poisoned samples are generated from samples whose clean labels are not equal to the attacker-chosen label.

After robustifying all the BadNets using (14), the results are reported in Table 8. In this table, CA and ASR of the BadNets are reported in the first two columns from the left. The third and fourth columns show our method's CA and ASR of the fine-tuned network. Also, to have a fair comparison with Neural Cleanse, CA and ASR of the fine-tuned networks using Neural Cleanse have been reported in the last two columns.

It can be seen that the robustification of the BadNets using our approach is successful in reducing the ASRs to 0 in three cases and 1.6% for Badnet-CR, which has a more complex trigger pattern. Moreover, it can be seen that after fine-tuning the network, the CA was not affected significantly, and the maximum reduction was 3.44%. Our method outperforms Neural Cleanse by achieving better ASRs. Specifically, Neural Cleanse is not successful for Badnet-GY, in which the trigger position is not fixed, and Badnet-YS, in which the trigger size is not small.

6 RAID—An On-line Detection Method

This section discusses the detection method called Removing Adversarial backdoors by Iterative Demarcation (RAID). The key differences between RAID and the previous method discussed in Sect. 5 are: (1) The previous method is purely off-line, whereas RAID uses both on-line and off-line models. (2) The previous method removes backdoors during off-line fine-tuning, whereas RAID rejects poisoned inputs during on-line implementation. (3) The previous method belongs to the reverse-engineering-based approach, whereas RAID belongs to the novelty-detection-based approach. RAID takes advantage of the on-line streaming data and therefore reduces reliance on off-line validation data and restrictive assumptions on triggers. While RAID may perform ineffectively initially because the on-line data are scarce at the beginning, its performance improves on-line over time and enables accurate detection of poisoned inputs as the on-line data are accumulated.

RAID first collects suspicious on-line samples that are highly likely to be poisoned using novelty detection models, which are trained off-line. Among the collected samples, RAID uses an anomaly detector to select the samples that are more likely to be poisoned than others. RAID trains a binary classifier using these selected data and the clean validation data. The trained binary classifier is used to determine if an on-line input is poisoned or not. As more suspicious on-line samples are collected, the binary classifier is updated, and the performance is improved. This repeated update to improve backdoor detection accuracy can be viewed as iterative demarcation. RAID uses two dimension-reduction methods to simplify the computational complexity. One is a feature extractor that compresses raw data into low-dimensional signature features, and the other is a dimension-reduction function such as PCA [53] that further reduces the feature dimensions. These two dimension-reduction methods result in a reduction of computational complexity, ensuring that RAID can be implemented in real time.

6.1 Problem Formulation

The scenario considered by RAID is as follows: the user outsources a training task to a third party, which returns a backdoored DNN $\mathcal{F}_b(.;\theta)$ (written for notational brevity simply as $\mathcal{F}_b$) to the user. The defender needs to build a detection model to protect the user from backdoor attacks. The attacker's goal is to make $\mathcal{F}_b$ have high CA and ASR. The defender's goal is to build the detection model $g(\cdot)$ to lower ASR of $\mathcal{F}_b$ while maintaining the CA. Mathematically, the defender wants $g(\cdot)$ to output "clean" with a high likelihood for clean samples x and "poisoned" with a high likelihood for poisoned samples x^*, i.e., to satisfy (6) and (7).

The attacker is considered to have complete control over the training dataset and process. However, the attacker neither has access to the user's validation dataset nor can change the model structure after training. The defender is considered to have a small set of clean validation data V (e.g., around 2% of the training dataset size). The defender has no prior information about the triggers or attacker-chosen label(s).

6.2 Detection Algorithm

The development of RAID was inspired by the observation that a DNN can be decomposed as a feature extractor C_b and a decision function $\mathcal{G}_b$, i.e., $\mathcal{F}_b = \mathcal{G}_b \circ C_b$ [54]. C_b reduces the raw data dimension and extracts features at higher abstraction levels. $\mathcal{G}_b$ maps the combinations of the features into corresponding outputs. The hypothesis is that the backdoor effect is created through a "logic" component encoded in $\mathcal{G}_b$ (i.e., specifying what features in the trigger should result in the attacker-chosen labels). RAID sets the output layer of $\mathcal{F}_b$ as $\mathcal{G}_b$ and all the previous layers as C_b.

The overall algorithm of RAID is shown in Algorithm 2. In **Given**, the defender has a backdoored network $\mathcal{F}_b$ decomposed into C_b and $\mathcal{G}_b$ and a small clean validation dataset V with the corresponding labels L. In **off-line training**, the defender obtains validation data features by feeding samples of V into C_b. Then, a new classifier $\mathcal{G}_n$ is trained with the extracted features V_F and the corresponding labels L. Since $\mathcal{G}_n$ is trained only on clean data, it is highly likely that $\mathcal{G}_n \circ C_b$ and $\mathcal{G}_b \circ C_b$ behave similarly on clean inputs but differently on poisoned inputs. A preprocess function (e.g., PCA) further reduces the feature dimension to obtain V_{FP}. V_{FP} is used to train a novelty detector $\mathcal{N}$ independent of $\mathcal{G}_n$ to detect inputs whose dimension-reduced features differ from dimension-reduced features of the clean validation data.

The on-line detection and retraining are shown in both Fig. 5 and **On-line Detection and Update** in Algorithm 2. The on-line implementation is comprised of a front-end part and a back-end part that may be implemented in a parallel manner. In the front-end part, C_b takes x as the input and outputs x_F. The preprocess function takes x_F as the input and outputs x_{FP}. If $g(x_{FP}) = 0$, i.e., if x is classified to be clean, then $\mathcal{F}_b(x)$ will be trusted. Otherwise, $\mathcal{F}_b(x)$ should not be trusted. $g(\cdot)$ is initialized as $g(x) = 0$ for all x and is then updated at a pre-specified frequency. In the back end, $\mathcal{N}$ and $\mathcal{G}_n$ determine if x is a "suspected" poisoned sample. If either $\mathcal{N}(x_{FP}) = 1$ (i.e., detected as different from the clean validation data) or $\mathcal{G}_n(x_F) \neq \mathcal{G}_b(x_F)$, then x will be collected into the anomalous dataset $\mathcal{A}$. $\mathcal{A}$ may

Algorithm 2 On-line detection algorithm of RAID

```
Given
  Validation data = $(V, L)$ and backdoored network $\mathcal{F}_b = \mathcal{G}_b \circ C_b$

Off-line Training
  Extract features of validation data: $V_F \leftarrow C_b(V)$
  Train a new classifier: $\mathcal{G}_n \leftarrow train(\mathcal{G}_n, (V_F, L))$.
  Reduce feature dimension: $V_{FP} \leftarrow preprocess(V_F)$
  Train a novelty detector: $\mathcal{N} \leftarrow train(\mathcal{N}, V_{FP})$

On-line Detection and Update
  $g(\cdot) \equiv 0$ (clean), count = 0, $\mathcal{A} = \{\}$, and determine window_size = $w_0$
  while True do
      count = count + 1
      Receive New Input $x$

      ### Make a Prediction on $x$ with $\mathcal{F}_b$ ###
      Extract Input Feature: $x_F \leftarrow C_b(x)$
      Make a Prediction: $l \leftarrow \mathcal{G}_b(x_F)$

      ### Check If $x$ is Poisoned (Front End) ###
      Reduce Feature Dimension: $x_{FP} \leftarrow preprocess(x_F)$
      if $g(x_{FP}) == 0$ then
          $l$ is the label for $x$ (Clean with High Probability)
      else
          $l$ is not the true label for $x$ (Poisoned with High Probability)
      end if

      ### Determine If $x$ Should Be Collected as Anomalous Data and Used to Update $g(\cdot)$ (Back
  End) ###
      $l' \leftarrow \mathcal{G}_n(x_F)$
      if $l \neq l'$ or $\mathcal{N}(x_{FP}) == 1$ then
          Collect $x_{FP}$: $\mathcal{A} \leftarrow \mathcal{A} \cup \{x_{FP}\}$
      end if

      ### Updating $g(\cdot)$ ###
      if count % window_size == 0 then
          Purify $\mathcal{A}$: $\mathcal{A}^* \leftarrow purify(\mathcal{A})$
          Update the Binary Classifier: $g \leftarrow train(g, (\{\mathcal{A}^*, 1\} \cup \{V_{FP}, 0\}))$
      end if
  end while
```

contain a few false positives. Therefore, an anomaly detector is used to purify $\mathcal{A}$ to obtain $\mathcal{A}^*$. $g(\cdot)$ is trained from scratch using V_{FP} and $\mathcal{A}^*$ at a pre-specified frequency. The defender can modify the window size (window_size in Algorithm 2) to determine the update frequency. Data in $\mathcal{A}^*$ is labeled as poisoned, and data in V_{FP} is labeled as clean.

$\mathcal{G}_n$ is a neural network with at most two hidden layers. This allows the structure of $\mathcal{G}_n \circ C_b$ to be close to the original backdoored network $\mathcal{F}_b = \mathcal{G}_b \circ C_b$. Therefore, they may show similar behavior on clean samples. Additionally, with such an architecture, the number of training parameters of $\mathcal{G}_n$ is small, reducing the clean

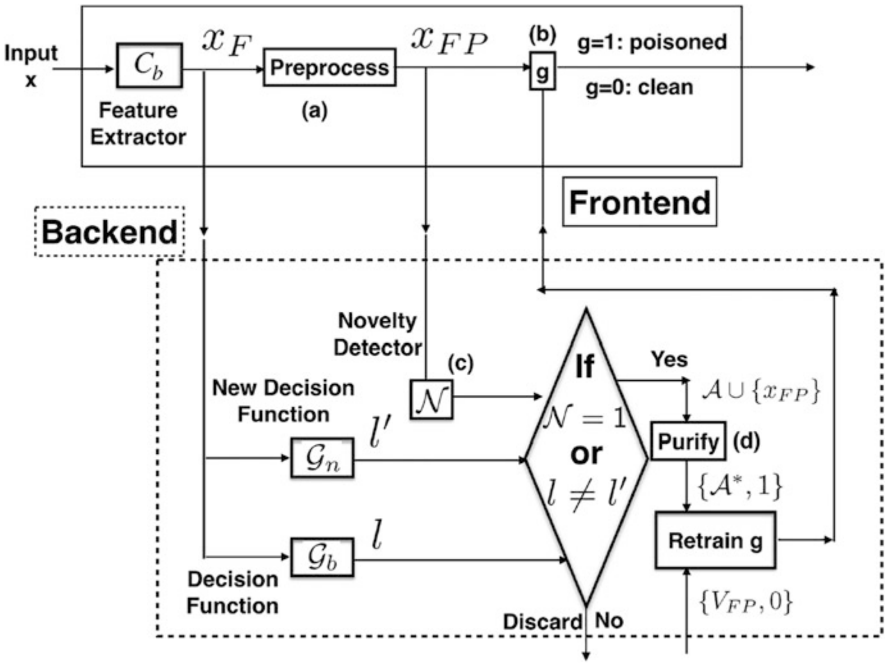

Fig. 5 Pipeline of the detection algorithm (on-line part)

validation samples required to train $\mathcal{G}_n$. In real-world cases, the defender may only have a small clean validation data. RAID uses SGD as the optimizer with a learning rate set to 0.01 and a cross-entropy-based loss function with default hyperparameter settings. PCA with the top 40 principals from the validation data features is used as the preprocess function (i.e., (a) in Fig. 5); hence, it reduces the data dimension without losing too much information and amplifies the spectral signature of the clean data. However, other dimension-reduction functions can also be considered, such as SVD and factor analysis from scikit-learn [55]. The PCA model in RAID cannot highlight the spectral signature of the triggers because it was trained only on a clean validation dataset.

Local outlier factor (LOF) is used as the novelty detector $\mathcal{N}$ (i.e., (c) in Fig. 5) since the training process is unsupervised and in real time. The anomaly detector (d) in the figure is also LOF for the same reasons. However, other outlier detectors (e.g., [56–59]) may be considered if the training process is also unsupervised and in real time. RAID uses SVM as the binary classifier $g(\cdot)$ in (b) in the figure. $g(\cdot)$ must be simple so that the training time is short. SVM satisfies this requirement. Other models, such as a neural network, may take a much longer training time than the SVM. However, the binary classifier is also open to other models that can be trained in real time.

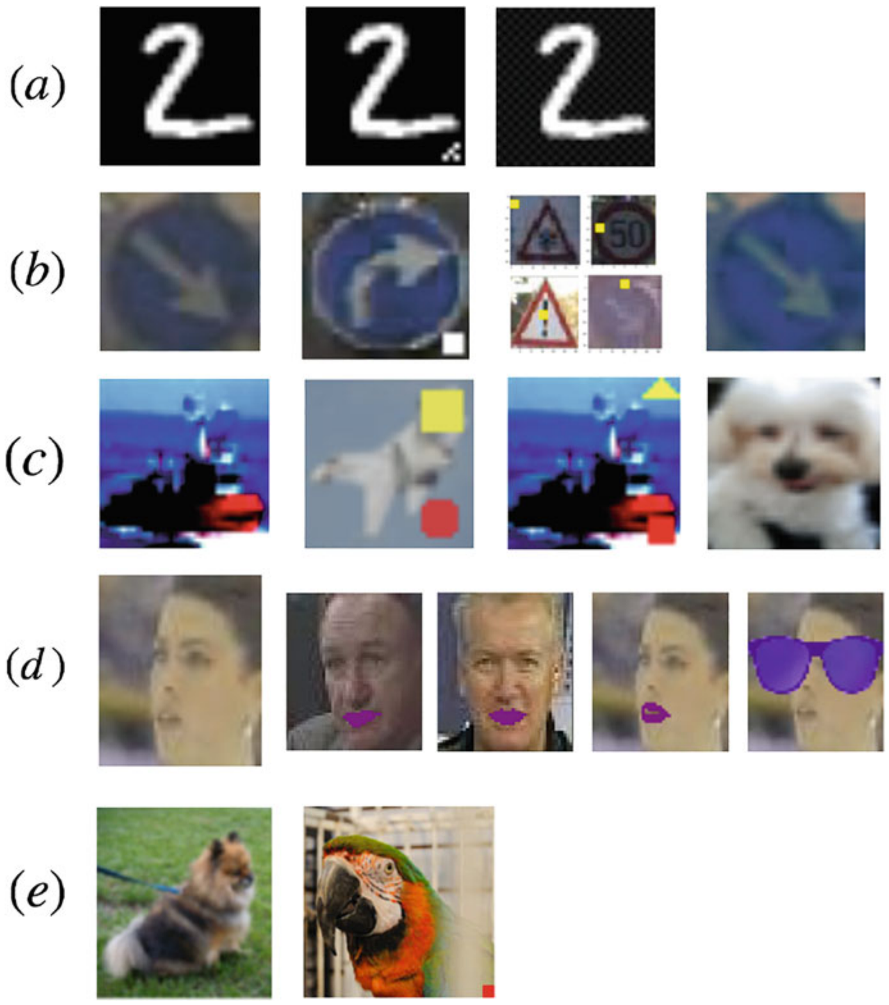

Fig. 6 The first column shows sample clean images for all the datasets. The other columns show all the triggers used in the experiments

6.3 Experimental Setup

RAID was implemented on five popular datasets: MNIST [60], German Traffic Sign Benchmarks (GTSRB) [48], CIFAR-10 [46], YouTube Face [50], and ImageNet [61]. Figure 6 shows different triggers used in the experiment. Several network architectures are used based on related works [18, 31, 47, 62–64]. The original paper of RAID [41] lists all the network architectures. Each dataset contains three

Table 9 Dataset Size. Columns 2–4 show the average number of samples per class

Dataset	Train	Valid	Test	Valid/train	# of classes
MNIST	5500	100	1000	1.8%	10
GTSRB	820	50	268	6.0%	43
CIFAR-10	5000	30	500	0.6%	10
YouT. Face	81	3	10	3.7%	1283
ImageNet	1200	3	25	0.2%	1000

Table 10 $\mathcal{F}_b$'s architecture for case (**a**)

Layer	Channels	Filter size	Stride	Activation
Conv2d	16	5×5	1	ReLU
MaxPool	16	2×2	2	–
Conv2d	32	5×5	1	ReLU
MaxPool	32	2×2	2	–
Linear	512	–	–	ReLU
Linear	10	–	–	–

parts: training part (Train), validation part (Valid), and testing part (Test), as shown in Table 9. Considering that the MNIST, GTSRB, and ImageNet datasets are unbalanced among classes, Table 9 reports the average number of samples per class, which is the dataset size divided by the total number of classes. The training part was partially poisoned (i.e., around 10%) and used for training the backdoored networks. The clean validation part was used for training $\mathcal{N}$ and $\mathcal{G}_n$. The testing dataset was used for evaluating RAID. The poisoned samples are acquired by injecting the triggers into the clean testing images. An on-line dataset with size n and attack density p is obtained with $(1-p)n$ clean testing samples plus pn poisoned testing samples. Empirically, when 20% of the test data is poisoned, RAID achieves low ASR and high CA. There is not much gain after $p = 0.5$.

6.3.1 BadNet Trained on MNIST

*Case (**a**)* The second column of Fig. 6a shows the trigger pattern. The attacker-chosen label l^* was determined by the ground-truth label l:

$$l^* = (l + 1) \bmod 10. \tag{15}$$

The network architecture is shown in Table 10. CA is 97.65%, and ASR is 96.3%.

*Case (**b**)* The trigger is shown in the third column in Fig. 6a, which is a dotted background and almost imperceptible. l^* is 0. The backdoored network has 89.35% CA and 100.0% ASR.

6.3.2 BadNet Trained on GTSRB

*Case (**c**)* The second column in Fig. 6b shows the trigger, which is a white box pattern. l^* is 33. CA is 96.41%, and ASR is 97.62%.

*Case (**d**)* The trigger is a moving square as shown in the third column in Fig. 6b with $l^* = 0$. CA is 95.26%, and ASR is 99.92%.

*Case (**e**)* Images passing through a Gotham filter will trigger $\mathcal{F}_b$, as shown in the fourth column in Fig. 6b. $l^* = 35$. CA is 94.49%, and ASR is 90.32%.

6.3.3 BadNet Trained on CIFAR-10

*Case (**f**)* The second picture in Fig. 6c shows the trigger, a combination of a box and a circle. $\mathcal{F}_b$ will output the attacker-chosen label 0 only when both the shapes appear on the input. CA is 88.6%, and ASR is 99.8%.

*Case (**g**)* The trigger is the combination of a triangle and a square, as shown in the third column of Fig. 6c. $l^* = 7$. CA is 88.83%, and ASR is 99.97%.

The last column in Fig. 6c shows another trigger, a small perturbation (one pixel at each corner). l^* is 0, CA is 82.44%, and ASR is 91.92%.

6.3.4 BadNet Trained on YouTube Face

*Case (**h**)* The trigger is sunglasses, as shown in the last column in Fig. 6d. l^* is 0. CA is 97.83%, and ASR is 99.98%.

*Case (**i**)* The trigger is red lipstick, as shown in the second column of Fig. 6d. l^* is 0. CA is 97.19%, and ASR is 91.43%.

*Case (**j**)* $\mathcal{F}_b$ has all three triggers: lipstick, eyebrow, and sunglasses, as shown in Fig. 6d. l^* is 4 for all the triggers. CA is 96.13%, and ASRs are 91.80%, 91.88%, and 100% on lipstick, eyebrow, and sunglasses.

*Case (**k**)* $\mathcal{F}_b$ has all three triggers as well. l^*, however, is 1 for lipstick, 5 for eyebrow, and 8 for sunglasses. CA is 96.08%, and ASRs are 91.11%, 91.10%, and 100% on lipstick, eyebrow, and sunglasses.

6.3.5 BadNet Trained on ImageNet

The trigger is a red box shown in the second column in Fig. 6e. $l^* = 0$. The network is DenseNet-121 [64]. CA is 72.14%, and ASR is 99.99%. This backdoor attack is used to evaluate the performance of RAID with different attack frequencies and validation dataset sizes.

6.3.6 Hyperparameter Setting

The contamination ratio is a hyperparameter defined in the LOF anomaly detector. A high contamination ratio means that the LOF will remove more samples, whereas a low contamination ratio means that the LOF will remove fewer samples. The contamination ratio was set to 0.2 for all the cases. Note that the contamination ratio is set by the defender. Thus, it is not equal to the proportion of outliers in the dataset. The proportion of outliers in the dataset is determined by the attacker, which is similar to the attack density p mentioned earlier. The number of neighbors (i.e., NN, which is another hyperparameter of LOF) was set to be 1. However, the defender can set any other values between 1 and 20. Empirically, it is observed that the CA and ASR of RAID with NN $=$ 1, 10, and 20 are comparable to each other, and there is not a distinctive advantage in choosing a higher NN; however, the lower NN makes the computation faster. The remaining parameters are set to typical default values.

6.4 Experimental Results

6.4.1 Performance of $\mathcal{N}$ and $\mathcal{G}_n$

Using both $\mathcal{N}$ and $\mathcal{G}_n$ can maximally identify poisoned samples. Table 11 shows the CA and ASR of using $\mathcal{N}$ alone, $\mathcal{G}_n$ alone, and both. Three dimension-reduction functions (i.e., PCA, TruncatedSVD, and FactorAnalysis from scikit-learn [55]) are utilized to train different $\mathcal{N}$. Since $\mathcal{G}_n$ does not use any dimension-reduced features, the three cases use the same $\mathcal{G}_n$. From the table, there is not much difference in the performance of $\mathcal{N}$ using PCA and SVD. Using FactorAnalysis leads to a higher ASR in some cases than the other two functions. However, we will see that the overall performance of $g(\cdot)$ using FactorAnalysis is also good. Additionally, the three functions are all fast enough for on-line usage. Therefore, if only $g(\cdot)$'s performance is considered, the three functions are almost equal, whereas if both $g(\cdot)$'s performance and $\mathcal{N}$'s performance are considered, PCA and TruncatedSVD provide better performance than FactorAnalysis. The performance of PCA and TruncatedSVD is almost equivalent. The table also shows that using $\mathcal{N}$ and $\mathcal{G}_n$ together will maximally catch poisoned samples (i.e., reduce ASR) and also increase false positives (i.e., reduce CA). However, in this initial filtering, the ASR should be weighed more than the CA. Additionally, an anomaly detector is used subsequently to reduce false positives further. In case (**e**), the ASR is still large even though $\mathcal{G}_n$ and $\mathcal{N}$ are used together because the trigger is more subtle. Rather than some specific patterns, case (**e**)'s trigger is a Gotham filter function. $\mathcal{G}_n$ and $\mathcal{N}$ are not sensitive to this trigger. But we will next show that RAID can still improve itself to reduce the ASR with on-line data further.

Table 11 Performance of $\mathcal{N} + \mathcal{G}_n$ on cases (a)–(i) for different dimension-reduction functions

	Classifier		PCA				TruncatedSVD				FactorAnalysis			
	$\mathcal{G}_n$		$\mathcal{N}$		$\mathcal{G}_n + \mathcal{N}$		$\mathcal{N}$		$\mathcal{G}_n + \mathcal{N}$		$\mathcal{N}$		$\mathcal{G}_n + \mathcal{N}$	
Case	CA	ASR	CA	ASR	CA	ASR	CA	ASR	CA	ASR	CA	ASR	CA	ASR
(a)	93.63	15.04	95.37	49.75	92.44	4.34	95.37	49.36	92.71	5.3	94.98	74.55	92.14	5.74
(b)	87.68	1.83	87.58	0.0	86.23	0.0	87.59	0.0	86.24	0	86.51	0	85.22	0
(c)	95.08	4.22	94.83	3.45	93.38	1.14	94.83	3.53	93.68	0.79	77.46	0.07	76.87	0.04
(d)	93.40	71.45	86.72	0	85.43	0	86.65	0	85.36	0	86.28	0	85.01	0
(e)	94.05	74.63	92.17	57.75	91.57	45.65	92.16	56.69	91.59	44.68	91.37	66.68	90.76	55.26
(f)	81.28	99.6	88.26	0.23	81.04	0.23	88.26	0.21	81.7	0.2	87.14	0.28	80.74	0.27
(g)	82.86	70	88.72	8.14	82.72	3.7	88.72	5.28	82.72	2.12	87.22	52.94	81.3	37.5
(h)	55.74	3.14	96.57	79.15	55.26	3.04	95.46	76.24	53.89	0.42	90.44	59.59	51.26	0.38
(i)	60.64	0.01	96.04	33.78	60.09	0.01	96.11	30.36	60.11	0.01	94.38	10.67	59.33	0.01

6.4.2 Performance of $g(\cdot)$

The first 40% of test data was used for on-line implementation and updating of the SVM, and the remaining 60% of test data was used for evaluating the performance of the backdoored network after employing the SVM. The binary SVM was initialized to output clean for all the inputs. Then, it was updated with a fixed window size, which is set to 10% of the test dataset size (therefore, the SVM can be updated 4 times in the considered test scenario). Each test input has an equal probability of being clean or poisoned. Table 12 shows the performance of RAID with PCA, SVD, and FactorAnalysis as the dimension-reduction function. From the table, RAID is effective with all the three dimension-reduction functions. Additionally, SVM helps reduce ASR while retaining high CA. Note that in case (**e**), the SVM still provides good performance, although the off-line models have high ASR (refer to Table 11). These results highlight the robustness of RAID. Table 13 lists how many poisoned samples are fed into the backdoored network and the size of $\mathcal{A}^*$ for training the SVM at each update when PCA was used as the dimension-reduction function. From the table, training a good SVM needs only a small set of poisoned samples. Since the performance of RAID using PCA, SVD, and FactorAnalysis is similar, we will only discuss the case when PCA is used as the dimension-reduction function for the following experiments.

6.4.3 Performance of the Anomaly Detector

We examine the performance of RAID with different contamination ratios. Table 14 shows the results. The classification accuracy drops significantly without the anomaly detector (i.e., the contamination ratio is 0). This is because the LOF does not remove any samples. Thus, $\mathcal{A}$ contains many false positives. When contamination ratio is 0, $\mathcal{A} = \mathcal{A}^*$. The SVM trained with such $\mathcal{A}^*$ will perform inefficiently. RAID shows comparable results when the contamination ratio is set to 0.1, 0.2, and 0.3. Both CA and ASR decrease, while the contamination ratio increases.

6.4.4 Multiple Triggers and Adaptive Attacks

The attacker may use multiple triggers to attack the backdoored network, i.e., cases **(j)** and **(k)**. The following attack scenarios are considered during on-line operation: (1) The attacker does not use any triggers. (2) The attacker uses only one of the triggers. (3) The attacker uses two of the triggers. (4) The attacker uses all the triggers. Scenario (1) is used for evaluating the performance of RAID when $\mathcal{A}^*$ only contains false positives. One can also consider scenario (1) as the case to evaluate RAID on a benign model. Scenarios (2), (3), and (4) are utilized to evaluate if RAID is effective with multiple triggers. The 4th updated SVM was tested. The results are shown in Table 15. From the table, RAID maintains high CA in all the scenarios and

Table 12 Performance of the backdoored network after using SVM with different dimension-reduction functions and after different numbers of updates (i.e., after 1st, 2nd, 3rd, and 4th updates)

BadNet			PCA				TruncatedSVD				FactorAnalysis			
Case		0th	1st	2nd	3rd	4th	1st	2nd	3rd	4th	1st	2nd	3rd	4th
(a)	CA	97.65	97.63	97.53	97.06	96.83	97.48	97.08	96.50	96.08	97.5	96.98	96.8	96.13
	ASR	96.3	38.4	11.35	3.38	1.15	9.66	2.25	0.76	0.5	9	2.71	1.96	1.21
(b)	CA	89.35	89.35	89.35	89.35	89.19	89.35	89.35	89.35	89.29	89.27	89.32	89.09	88.69
	ASR	100	0	0	0	0	0	0	0	0	0	0	0	0
(c)	CA	96.41	96.41	96.41	96.41	96.41	96.41	96.41	96.41	96.41	96.41	96.41	96.41	96.37
	ASR	97.62	1.17	0.97	0.29	0.26	0.69	0.54	0.42	0.34	0.91	0.5	0.35	0.35
(d)	CA	95.26	95.19	94.76	94.57	94.61	94.37	94.47	94.40	94.37	95.19	95.03	94.58	94.60
	ASR	99.92	0.55	0.23	0.21	0.19	0.14	0.18	0.13	0.09	0.7	0.26	0.22	0.14
(e)	CA	94.49	94.37	93.87	93.52	93.36	94.47	94.02	93.56	93.41	94.49	94.40	93.70	93.04
	ASR	90.32	20.26	10.16	7.44	4.61	25.94	12.35	8.81	5.7	45.31	24.25	10.09	2.75
(f)	CA	88.6	88.6	88.6	88.6	88.6	88.6	88.6	88.6	88.6	88.6	88.6	88.5	88.5
	ASR	99.88	0.05	0.01	0.03	0.01	0.03	0.03	0.03	0	0.03	0.01	0	0
(g)	CA	88.83	88.83	88.83	88.83	88.83	88.83	88.83	88.83	88.83	88.83	88.83	88.83	88.83
	ASR	99.7	0.4	0.2	0.13	0.1	0.4	0.2	0.2	0.06	3.1	0.7	0.4	0.2
(h)	CA	97.83	97.81	97.77	97.73	97.67	97.83	97.80	97.79	97.79	97.80	97.73	97.71	97.68
	ASR	99.98	8.96	2.84	1.07	0.48	6.69	3.02	1.23	0.53	2.53	1.54	0.61	0.42
(i)	CA	97.19	97.19	97.16	97.16	97.11	97.18	97.15	97.14	97.14	97.18	97.11	97.11	97.11
	ASR	91.43	6.67	4.35	3.83	3.24	6.17	3.96	3.27	2.97	4.98	3.70	2.84	2.53

Table 13 Size of $\mathcal{A}^*$ and the numbers of poisoned inputs that have appeared at each update

		0th	1st	2nd	3rd	4th
(a)–(b)	# of poi.	0	1000	2000	3000	4000
(a)	size of $\mathcal{A}^*$	0	217	411	609	812
(b)		0	226	439	666	894
(c)–(e)	# of poi.	0	1263	2526	3789	5052
(c)	size of $\mathcal{A}^*$	0	275	534	779	1033
(d)		0	295	575	870	1123
(e)		0	162	303	434	571
(f)–(g)	# of poi.	0	1000	2000	3000	4000
(f)	size of $\mathcal{A}^*$	0	213	424	635	846
(g)		0	214	412	614	826
(h)–(i)	# of poi.	0	1283	2566	3849	5132
(h)	size of $\mathcal{A}^*$	0	360	722	1086	1453
(i)		0	352	698	1044	1390

Table 14 Performance of RAID (the 4th update) with different contamination ratios

	Ratio = 0.0		Ratio = 0.1		Ratio = 0.2		Ratio = 0.3	
	CA	ASR	CA	ASR	CA	ASR	CA	ASR
(a)	86.35	0.03	96.90	1.7	96.83	1.15	94.48	0.15
(b)	87.69	0	89.35	0	89.19	0	89.02	0
(c)	92.57	0.05	94.41	0.65	94.41	0.26	94.34	0.23
(d)	94.24	0.06	95.72	0.25	94.61	0.19	94.47	0.10
(e)	88.53	0.19	94.04	13.77	93.36	4.61	93.0	2.42
(f)	56.63	0	88.6	0.05	88.6	0.01	88.6	0.01
(g)	62.03	0	88.83	0.13	88.83	0.1	88.83	0.1
(h)	92.64	0	97.76	1.81	97.67	0.48	97.60	0.24
(i)	92.23	0.85	97.19	4.94	97.11	3.24	97.07	2.58

has low ASR on triggers that the attacker has used. For scenario (1), although $\mathcal{A}^*$ contains only false positives, RAID still manages to have a high CA. For scenarios (2) and (3), RAID has high ASR on the second or third trigger. However, since the SVM is updated in real time, once the new triggers are used for backdoor attacks, $\mathcal{N}$ and $\mathcal{G}_n$ will detect them in the back end resulting in attack detection by the SVM in the next update, such as case (4). The only period in which the network is vulnerable to the new triggers is between the moment that a new trigger appears and the next SVM update. Overall, the results show the efficacy and robustness of RAID.

6.4.5 Experiments on Hyperparameters

The first experiment is to evaluate RAID with different numbers of principal components. A backdoored network with small perturbations (only one pixel at each corner) as the trigger (i.e., the third $\mathcal{F}_b$ in the CIFAR-10 case) was trained, which has

Table 15 RAID performance on dynamic attacks (j)–(k)

Case/attack		Net.	(1)	(2)	(3)	(4)
(j)	CA	96.13	96.10	96.05	95.83	95.88
	ASR1	91.80	91.78	3.84	3.72	3.79
	ASR2	91.88	91.85	64.36	2.22	2.27
	ASR3	100	100	100	100	0.21
(k)	CA	96.08	96.05	95.99	95.90	95.88
	ASR1	91.11	91.11	2.77	2.84	3.21
	ASR2	91.10	91.10	89.88	0.99	0.94
	ASR3	100	100	99.65	95.82	0

82.44% CA and 91.92% ASR. The first two pictures in Fig. 7 show the performance of RAID with different numbers of principal components. From the pictures, all the plots show significant drops in ASR at different rates. Using more principal components results in a faster reduction in ASR. Using fewer principal components results in a small drop in CA (i.e., around 1%). With fewer principal components, the dimension-reduced poisoned data features are closer to the dimension-reduced clean data features. Thus, $\mathcal{A}^*$ may contain more false positives, which increases the training noise and leads to the degradation of CA. The number of principal components should be from 20% to 40% of the original feature dimension.

The second experiment is to evaluate RAID with a different attack frequency/-density (i.e., the probability p of an input being poisoned). Note that the attack frequency/density is determined by the attacker. Therefore, it is different from the contamination ratio. We tested RAID on the ImageNet dataset because we also want to see if RAID is efficient on a large-scale dataset. The backdoored model is DenseNet-121 with 72.14% CA and 99.99% ASR. The dataset and trigger are shown in Fig. 6e. The middle two pictures in Fig. 7 show the effectiveness of RAID on ImageNet under different attack densities. It is seen that ASR reduces faster when attack density is higher (i.e., more poisoned inputs are fed into the network). When attack density is 0 (meaning $\mathcal{A}^*$ has only false positives), the CA is still high.

RAID was also tested with 1 or 2 images per class on ImageNet to see if the clean validation dataset could be even smaller. Although low ASR is achieved with one image per class, CA degrades (the last two pictures in Fig. 7). This is because the novelty detector $\mathcal{N}$ and the new classifier $\mathcal{G}_n$ generate many false positives due to a lack of training data. Therefore, $\mathcal{A}^*$ may contain many false positives, which increases the training noise and leads to the degradation of CA (i.e., the SVM is trained with bad training samples).

The last experiment is to evaluate the performance of RAID when the SVM is updated at different frequencies. During the period between two updates, the backdoored network might be exposed to an attack if new triggers are applied. Therefore, reducing this period (increasing update frequency) can help further mitigate the threat of new triggers. The user needs to set a window size for the update. For example, if the window size is 1000, the SVM will be updated once there are 1000 new inputs into the network. Figure 8 shows the performance of

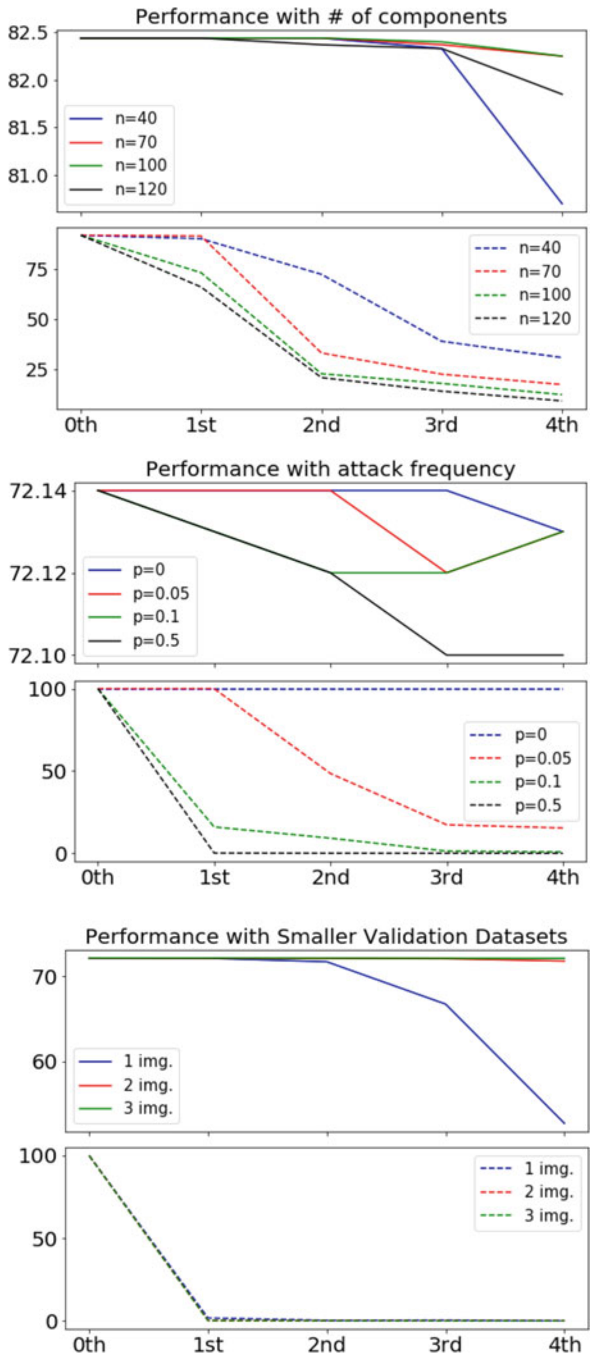

Fig. 7 Solid lines in all the pictures: CA. Dashed lines in all the pictures: ASR. X-axis: the number of updates. n: the number of PCA components. p: the probability of a sample being poisoned. For the last two pictures: the red plots overlap the other plots and are not visible

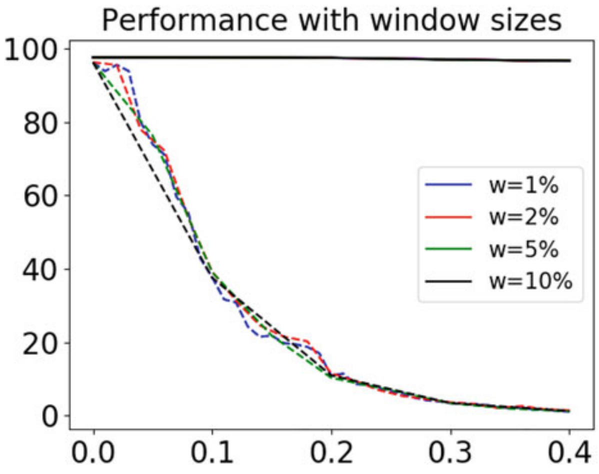

Fig. 8 X-axis: ratio of test data size used in RAID to the total test data size. Solid lines: CA. Dashed lines: ASR. w: window size/test data size

RAID with different update frequencies under case (**a**). RAID shows consistently good performance after increasing the update frequency (i.e., reducing window size). Since training the SVM is quick (<1 s), RAID can be used effectively during on-line operation.

6.4.6 More Advanced Attack

[20] propose a backdoor attack with sample-specific triggers. The example is shown in Fig. 9. It can be seen that the trigger remains invisible in the image. We use a subset of the ImageNet dataset as the testing data. The backdoored model has 78% CA and 100% ASR. The model architecture is ResNet-18 [51]. After the 4th update, RAID reduces the ASR to 0.4% and keeps CA close to 78%.

The attacker may try to minimize the feature-level outputs between poisoned data and corresponding clean data to bypass RAID. However, the difference between clean and poisoned data always exists and must be represented in hidden layer outputs. Otherwise, if the hidden layer outputs are identical for clean and poisoned samples, the network outputs should then also be the same. This cannot be true since the network outputs the attacker-chosen label for the poisoned sample and the ground-truth label for the clean sample. Although the difference may be small for one hidden layer, the cumulative difference for multiple hidden layers becomes significant and observable. RAID can still be applied by changing the input of its novelty detector and the binary classifier to include multiple hidden layer output features.

As seen above, RAID fuses several simple models (i.e., simple neural networks, novelty detector, anomaly detector, dimension-reduction function, and binary clas-

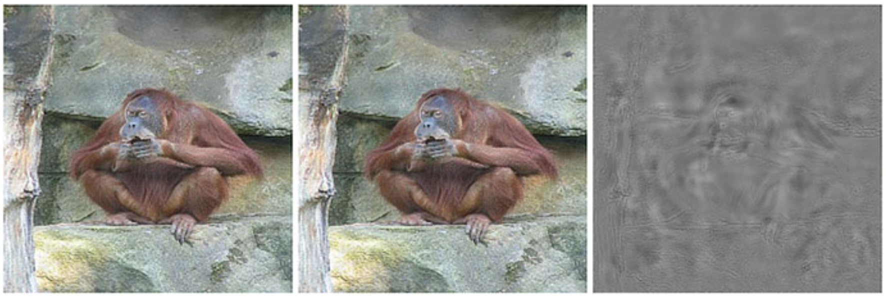

Fig. 9 Sample-specific trigger. Left: benign image. Middle: poisoned image. Right: the corresponding trigger

sifier) to reduce the ASR caused by the attacker. It requires only a small clean validation dataset, which is feasible to acquire in real-world applications.

7 Benign Applications of the Backdoor Phenomena

While we have considered backdoor-based attacks in this chapter, it is to be noted that backdoors can also be used for benign purposes such as the protection of intellectual properties. One example is using the backdoors as watermarks [65]. To train an accurate neural network model, the trainer needs to invest considerable cost and effort to collect high-quality data, label the data, buy/rent computational resources, and tune the model hyperparameters. Therefore, it is critical to find a way to protect the intellectual property of the models. Similar to watermark injection into documents, neural network models can also be injected with watermarks. Backdoor-based watermarks are one option for this purpose. The trainer injects backdoors into the trained network so that any other network copied based on this model can be recognized by presenting it with the poisoned inputs. Other models cannot correctly predict the outputs since they do not know the trigger information. The model's performance on clean samples, however, is not affected. Therefore, intellectual properties can be protected by utilizing backdoor attack mechanisms.

8 Future Directions

Although our methods introduced in this chapter utilize only a small clean validation dataset, it is of value to further decrease the size of the required validation dataset. RAID requires some on-line data to attain samples of possible poisoned inputs in addition to the validation dataset to train a classifier for clean vs. poisoned inputs. During this period, some poisoned inputs may escape detection. Reducing this transient vulnerability is therefore an avenue for future improvements. Additionally, the reduction of computational complexity is also an important topic for future work. Another extension would be to generalize the methods to backdoor detection for machine learning methods that are not based on neural networks. Further application of the methods to other adaptive triggers should also be considered. Lastly, using explainability tools for DNNs may be helpful to further improve the applicability and usability of backdoor detection methods.

References

1. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Proceedings of the Advances in Neural Information Processing Systems, pp. 1097–1105. Lake Tahoe, Nevada (2012)
2. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition (2014). arXiv preprint arXiv:1409.1556
3. Szegedy, C., Ioffe, S., Vanhoucke, V., Alemi, A.A.: Inception-v4, Inception-ResNet and the impact of residual connections on learning. In: Proceedings of the 31st AAAI Conference on Artificial Intelligence, San Francisco, pp. 4278–4284 (2017)
4. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: Proceedings of the 13th European Conference on Computer Vision, Zurich, Switzerland, pp. 818–833 (2014)
5. Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, A.R., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T.N., Kingsbury, B.: Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. IEEE Signal Process. Mag. **29**(6), 82–97 (2012)
6. Sainath, T.N., Mohamed, A.-R., Kingsbury, B., Ramabhadran, B.: Deep convolutional neural networks for LVCSR. In: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing. Vancouver, pp. 8614–8618 (2013)
7. Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., Khudanpur, S.: Recurrent neural network based language model. In: Proceedings of the 11th Annual Conference of the International Speech Communication Association. Chiba, Japan, pp. 1045–1048 (2010)
8. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Proceedings of the Advances in Neural Information Processing Systems, Lake Tahoe, pp. 3111–3119 (2013)
9. Fu, H., Krishnamurthy, P., Khorrami, F.: Functional replicas of proprietary three-axis attitude sensors via LSTM neural networks. In: Proceedings of the IEEE Conference on Control Technology and Applications. Montreal, pp. 70–75 (2020)
10. Chen, C., Seff, A., Kornhauser, A., Xiao, J.: DeepDriving: learning affordance for direct perception in autonomous driving. In: Proceedings of the IEEE International Conference on Computer Vision. Santiago, pp. 2722–2730 (2015)
11. Schwarting, W., Alonso-Mora, J., Rus, D.: Planning and decision-making for autonomous vehicles. Ann. Rev. Control Robot. Auton. Syst. **1**, 187–210 (2018)
12. Hadsell, R., Sermanet, P., Ben, J., Erkan, A., Scoffier, M., Kavukcuoglu, K., Muller, U., LeCun, Y.: Learning long-range vision for autonomous off-road driving. J. Field Robot. **26**(2), 120–144 (2009)
13. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: Proceedings of the International Conference on Learning Representations. San Diego, pp. 1–14 (2015)

14. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks (2013). arXiv preprint arXiv:1312.6199
15. Liu, Y., Ma, S., Aafer, Y., Lee, W.C., Zhai, J., Wang, W., Zhang, X.: Trojaning attack on neural networks. In: Proceedings of the 25th Annual Network and Distributed System Security Symposium. San Diego, pp. 18–221 (2018)
16. Gu, T., Dolan-Gavitt, B., Garg, S.: BadNets: identifying vulnerabilities in the machine learning model supply chain (2017). arXiv preprint arXiv:1708.06733
17. Liu, K., Dolan-Gavitt, B., Garg, S.: Fine-pruning: defending against backdooring attacks on deep neural networks. In: Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses. Heraklion, pp. 273–294 (2018)
18. Liu, K., Tan, B., Karri, R., Garg, S.: Poisoning the (data) well in ML-based CAD: a case study of hiding lithographic hotspots. In: Proceedings of the Design, Automation & Test in Europe Conference & Exhibition. Grenoble, pp. 306–309 (2020)
19. Wenger, E., Passananti, J., Bhagoji, A.N., Yao, Y., Zheng, H., Zhao, B.Y.: Backdoor attacks against deep learning systems in the physical world. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Virtual, pp. 6206–6215 (2021)
20. Li, Y., Li, Y., Wu, B., Li, L., He, R., Lyu, S: Invisible backdoor attack with sample-specific triggers. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, Virtual, pp. 16463–16472 (2021)
21. Saha, A., Subramanya, A., Pirsiavash, H.: Hidden trigger backdoor attacks. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34. New York, pp. 11957–11965 (2020)
22. Li, S., Xue, M., Zhao, B., Zhu, H., Zhang, X.: Invisible backdoor attacks on deep neural networks via steganography and regularization. IEEE Trans. Depend. Secure Comput. **18**(5), 2088–2105 (2020)
23. Liu, Y., Ma, X., Bailey, J., Lu, F.: Reflection backdoor: a natural backdoor attack on deep neural networks. In: Proceedings of the European Conference on Computer Vision, Virtual, pp. 182–199 (2020)
24. Xie, C., Huang, K., Chen, P.-Y., Li, B.: DBA: distributed backdoor attacks against federated learning. In: Proceedings of the International Conference on Learning Representations. New Orleans (2019)
25. Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D., Shmatikov, V.: How to backdoor federated learning. In: Proceedings of the International Conference on Artificial Intelligence and Statistics, Virtual, pp. 2938–2948 (2020)
26. Andreina, S., Marson, G.A., Möllering, H., Karame, G.: BaFFLe: backdoor detection via feedback-based federated learning. In: Proceedings of the IEEE International Conference on Distributed Computing Systems, Virtual, pp. 852–863 (2021)
27. Yao, Y., Li, H., Zheng, H., Zhao, B.Y.: Latent backdoor attacks on deep neural networks. In: Proceedings of the ACM SIGSAC Conference on Computer and Communication Security. London, pp. 2041–2055 (2019)
28. Zhang, Z., Jia, J., Wang, B., Gong, N.Z.: Backdoor attacks to graph neural networks. In: Proceedings of the ACM Symposium on Access Control Models and Technology, Virtual, pp. 15–26 (2021)
29. Dai, J., Chen, C., Li, Y.: A backdoor attack against LSTM-based text classification systems. IEEE Access **7**, 138872–138878 (2019)
30. Gong, X., Chen, Y., Wang, Q., Huang, H., Meng, L., Shen, C., Zhang, Q.: Defense-resistant backdoor attacks against deep neural networks in outsourced cloud environment. IEEE J. Sel. Areas Commun. **39**(8), 2617–2631 (2021)
31. 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: Proceedings of the 40th IEEE Symposium on Security and Privacy. San Francisco, pp. 707–723 (2019)
32. Guo, W., Wang, L., Xing, X., Du, M., Song, D.: TABOR: a highly accurate approach to inspecting and restoring trojan backdoors in AI systems (2019). arXiv preprint arXiv:1908.01763

33. Chen, H., Fu, C., Zhao, J., Koushanfar, F.: DeepInspect: a black-box trojan detection and mitigation framework for deep neural networks. In: Proceedings of the 28th International Joint Conference on Artificial Intelligence. International Joint Conferences on Artificial Intelligence Organization, Macao, pp. 4658–4664 (2019)
34. Xu, X., Wang, Q., Li, H., Borisov, N., Gunter, C.A., Li, B.: Detecting AI trojans using meta neural analysis (2019). arXiv preprint arXiv:1910.03137
35. Li, Y., Ma, H., Zhang, Z., Gao, Y., Abuadbba, A., Fu, A., Zheng, Y., Al-Sarawi, S.F. Abbott, D.: NTD: non-transferability enabled backdoor detection (2021). arXiv preprint arXiv:2111.11157
36. 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 ACM SIGSAC Conference on Computer and Communications Security. London, pp. 1265–1282 (2019)
37. Lee, K., Lee, K., Lee, H., Shin, J.: A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In: Proceedings of the Conference on Neural Information Processes Systems. Montreal, pp. 7167–7177 (2018)
38. Chou, E., Tramèr, F., Pellegrino, G., Boneh, D.: SentiNet: detecting physical attacks against deep learning systems (2018). arXiv preprint arXiv:1812.00292
39. 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. San Juan, pp. 113–125 (2019)
40. Kwon, H.: Detecting backdoor attacks via class difference in deep neural networks. IEEE Access **8**, 191049–191056 (2020)
41. Fu, H., Veldanda, A.K., Krishnamurthy, P., Garg, S., Khorrami, F.: A feature-based on-line detector to remove adversarial-backdoors by iterative demarcation. IEEE Access **10**, 5545–5558 (2022)
42. Chen, B., Carvalho, W., Baracaldo, N., Ludwig, H., Edwards, B., Lee, T., Molloy, I., Srivastava, B.: Detecting backdoor attacks on deep neural networks by activation clustering (2018). arXiv preprint arXiv:1811.03728
43. Tran, B., Li, J., Madry, A.: Spectral signatures in backdoor attacks. In: Proceedings of Advances in Neural Information Processing Systems, vol. 31, Montreal, pp. 8000–8010 (2018)
44. Tang, D., Wang, X., Tang, H., Zhang, K.: Demon in the variant: statistical analysis of DNNs for robust backdoor contamination detection. In: Proceedings of the 30th USENIX Security Symposium, Virtual, pp. 1541–1558 (2021)
45. Wang, Z., Simoncelli, E.P., Bovik, A.C.: Multiscale structural similarity for image quality assessment. In: Proceedings of the 37th Asilomar Conference on Signals, Systems & Computers, vol. 2, Pacific Grove, pp. 1398–1402 (2003)
46. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)
47. Lin, M., Chen, Q., Yan, S.: Network in network. In: Proceedings of the International Conference on Learning Representations, Banff, pp. 1–10 (2014)
48. Stallkamp, J., Schlipsing, M., Salmen, J., Igel, C.: The German Traffic Sign Recognition Benchmark: a multi-class classification competition. In: Proceedings of the International Joint Conference on Neural Networks. San Jose, pp. 1453–1460 (2011)
49. Veldanda, A.K., Liu, K., Tan, B., Krishnamurthy, P., Khorrami, F., Karri, R., Dolan-Gavitt, B., Garg, S.: NNoculation: broad spectrum and targeted treatment of backdoored DNNs (2020). arXiv preprint arXiv:2002.08313
50. Wolf, L., Hassner, T., Maoz, I.: Face recognition in unconstrained videos with matched background similarity. In: Proceedings of the IEEE Computer Vision and Pattern Recognition. Colorado Springs, pp. 529–534 (2011)
51. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Las Vegas, pp. 770–778 (2016)
52. Miller, J.: Reaction time analysis with outlier exclusion: bias varies with sample size. Q. J. Exp. Psychol. **43**(4), 907–912 (1991)

53. Tipping, M.E., Bishop, C.M.: Probabilistic principal component analysis. J. R. Stat. Soc. Ser. B (Statistical Methodology) **61**(3), 611–622 (1999)
54. Fu, H., Veldanda, A.K., Krishnamurthy, P., Garg, S., Khorrami, F.: Detecting backdoors in neural networks using novel feature-based anomaly detection (2020). arXiv preprint arXiv:2011.02526
55. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
56. Chen, Y., Zhou, X.S., Huang, T.S.: One-class SVM for learning in image retrieval. In: Proceedings of International Conference on Image Processing, vol. 1, Thessaloniki, pp. 34–37 (2001)
57. Dong, Y., Hopkins, S., Li, J.: Quantum entropy scoring for fast robust mean estimation and improved outlier detection. In: Proceedings of the Advances in Neural Information Processing Systems. Vancouver, pp. 6067–6077 (2019)
58. Hariri, S., Kind, M.C., Brunner, R.J.: Extended isolation forest. IEEE Trans. Knowl. Data Eng. **33**(4), 1479–1489 (2019)
59. Lesouple, J., Baudoin, C., Spigai, M., Tourneret, J.-Y.: Generalized isolation forest for anomaly detection. Pattern Recognit. Lett. **149**, 109–119 (2021)
60. LeCun, Y., Cortes, C.: MNIST handwritten digit database (2010). http://yann.lecun.com/exdb/mnist
61. Deng, J., Dong, W., Socher, R., Li, L., Li, K., Fei-Fei, L.: ImageNet: a large-scale hierarchical image database. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Miami, pp. 248–255 (2009)
62. Sun, Y., Wang, X., Tang, X.: Deep learning face representation from predicting 10,000 classes. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Columbus, pp. 1891–1898 (2014)
63. Gu, T., Liu, K., Dolan-Gavitt, B., Garg, S.: BadNets: evaluating backdooring attacks on deep neural networks. IEEE Access **7**, 47230–47244 (2019)
64. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Honolulu, pp. 4700–4708 (2017)
65. Adi, Y., Baum, C., Cisse, M., Pinkas, B., Keshet, J.: Turning your weakness into a strength: watermarking deep neural networks by backdooring. In: Proceedings of the 27th USENIX Security Symposium. Baltimore, pp. 1615–1631 (2018)