Variance-Based Defense Against Blended Backdoor Attacks

Sujeevan Aseervatham[1,3] (✉), Achraf Kerzazi[1,2,3], and Younès Bennani[2,3]

[1] Orange Research, Châtillon, France
{sujeevan.aseervatham, achraf.kerzazi}@orange.com
[2] Université Sorbonne Paris Nord, LIPN, UMR 7030 CNRS
younes.bennani@sorbonne-paris-nord.fr
[3] LaMSN - La Maison des Sciences Numériques, F-93210, Plaine Saint-Denis - France

Abstract. Backdoor attacks represent a subtle yet effective class of cyberattacks targeting AI models, primarily due to their stealthy nature. The model behaves normally on clean data but exhibits malicious behavior only when the attacker embeds a specific trigger into the input. This attack is performed during the training phase, where the adversary corrupts a small subset of the training data by embedding a pattern and modifying the labels to a chosen target. The objective is to make the model associate the pattern with the target label while maintaining normal performance on unaltered data.
Several defense mechanisms have been proposed to sanitize training datasets. However, these methods often rely on the availability of a clean dataset to detect statistical anomalies, which may not always be feasible in real-world scenarios where datasets can be unavailable or compromised. To address this limitation, we propose a novel defense method that trains a model on the given dataset, detects poisoned classes, and extracts the critical part of the attack trigger before identifying the poisoned instances. This approach enhances explainability by explicitly revealing the harmful part of the trigger. The effectiveness of our method is demonstrated through experimental evaluations on well-known image datasets and a comparative analysis against three state-of-the-art algorithms: SCAn, ABL, and AGPD.

Keywords: Machine Learning · Data Poisoning · Backdoor Mitigation

1 Introduction

The rise of Artificial Intelligence (AI), and more specifically Deep Learning-based systems, has been dazzling and unparalleled. Its use is now widespread and, thanks to the availability of free software, a large public, even without specific knowledge, can build an AI model. They can also download training datasets and pre-trained models. Like any software, AI systems are vulnerable to malicious attacks. They can traditionally be attacked at the software layer, e.g., through vulnerabilities in the software libraries, but they can also be attacked through data and model-parameter manipulation. A corrupted AI can exhibit malicious

behavior, which can have critical consequences. For example, self-driving cars may intentionally collide depending on the attacker's aim and a Large Language Model (LLM) may give fake news. This security risk has become a real-life threat and a major issue when researchers have shown that the AI models can be attacked in various ways depending on the aim, knowledge, and model access privilege of the attacker [16,21]. The attacker can lead an evasion attack by slightly modifying the model's input in order to change the decision/prediction. For example, for a spam recognition system, the attacker may change some words to bypass the filter. She can also lead inference attacks where the attacker wants to extract some information about the training dataset on which the model was trained. This attack is also known as privacy attack or model inversion attack. Indeed, in a personalized drug system, she may want to know if a particular patient information was in the training set. During the training stage, an attacker can also poison the training data to either degrade the model's performance (non-targeted attacks), e.g., by flipping the labels, or introduce a backdoor than can be exploited, during inference, to trigger a malicious behavior of the model.

In this paper, we focus on a defense method against the backdoor attacks where the attacker wants a model to associate her data pattern to her target label. Thus, in the inference stage, any input containing the pattern will trigger the misclassification of the input as the target label. This is a vicious attack as the model seems to be pristine w.r.t. to its performance on clean data but in fact contains a backdoor which changes the prediction only in the presence of the trigger in the input data. For example, in a self-driving car, an attacker may have corrupted the model to recognize a green-square sticker pasted on a stop sign as a 70mph speed limit board. Such attacks can easily be performed when the attacker has a write access to the training data, as illustrated by the BadNets algorithm [7]. More advanced attacks based on BadNets have been proposed, but they often require more resources or access rights. By its simplicity, BadNets remains a threat that can be exploited by a large population, including opportunistic criminals and script-kiddies (open source script users without particular knowledge). Many defenses have been proposed in the literature, but most of them require an additional dataset of clean data on which statistics are computed to detect a distribution anomaly induced by a poisoned sample. In real-life, such dataset may hardly be available and even be subject to a corruption.

We propose a multi-stage defense algorithm that estimates the poisoned subset of the training data, detects the poisoned classes, extracts the attack trigger for each class, and identifies the poisoned instances based on the extracted pattern. Our contributions can be summarized in three points: 1) we introduce a novel defense algorithm against BadNets [7] and Blended [2] attacks which does not rely on additional prerequisites such as the availability of a clean dataset; 2) the defense is explainable, as it extracts the most important part of the trigger responsible for the malicious behavior; and 3) our method is effective in All-to-All attacks where many classes are poisoned.

The remainder of this paper is organized as follows: in Section 2, we present the state-of-the-art of backdoor attacks and defenses. Our defense algorithm is detailed in Section 3. Section 4 describes the experimental results and, finally, Section 5 concludes this article.

2 Related Work

2.1 Backdoor Attacks

The Backdoor attack was introduced in [7] where the proposed BadNets methodology involves patching a small pattern onto the input data of a small percentage of the training set and labeling them with the desired target label. By training on this poisoned set, the model learns to associate the attack pattern with the target label. Thus, the presence of the pattern in an input instance triggers the malicious behavior of the model by classifying the input as the attacker's target label instead of the correct one. To make the pattern more stealthy, linear blending is used in [2], while in [29] the pattern is generated in the low-frequency domain by considering both the training set and a pre-trained model. These works paved the way for more advanced attacks, where the pattern is not fixed but specific to the input [19,18]. However, compared to fixed pattern attacks, input-based pattern attacks require more prerequisites, such as access to and modification of the training algorithm, making them more suitable when targeting users who download pretrained models. Fixed-pattern attacks remain more realistic in real-world scenarios, especially when the attacker is an insider with access to the training data.

2.2 Backdoor Defenses

In the last few years, many backdoor defense methods have been proposed by researchers, each designed to be used at a specific stage of the AI model lifecycle. As noted in [28] and [30], these defenses can be grouped in four categories based on the lifecycle stage: 1) pre-training stage methods, mainly used to detect the poisoned instances and sanitize the dataset [1,6,4,25,17,22,28], 2) in-training stage defenses, which aim to reduce the effect of the poisoned data on the model [14,10,30], 3) post-training stage methods, used to correct backdoored models [27,15], and 4) inference stage defenses, which are used to detect malicious inputs by analyzing both both the input and the output of the model, typically residing between the user and the model [8]. These methods can also be categorized based on the information they use to mitigate the attacks. The main approaches are input-based, loss-based, and activation-based. Defenses in the pre-training and inference stages are mainly input-based or activation-based methods, while algorithms in the in-training and post-training stages are mostly loss-based or activation-based. Input-based approaches operate on the input data by altering it and computing statistical measures on the predictions, such as the entropy [6,4,8]. In [6], the STRIP algorithm linearly blends the input image with a set

of clean images before computing the entropy of the prediction on each perturbed images. A poisoned image is detected when the entropy is low. In [4], the SentiNet method uses Grad-CAM [24] to extract the decision region from an input image and superimposes it on a set of clean images to check whether it triggers a misclassification of the model. Activation-based methods rely on the assumption that the poisoned and clean samples can be separated, e.g., through clustering, in the feature space induced by the activations of a specific layer of the model [1,25,17,22]. In [1], K-Means clustering is used on the activations and in [25], the proposed SCAn algorithm uses a two-component Gaussian Mixture Models (GMM) for clustering. In the loss-based approach, the methods rely on the property that poisoned instances are classified with a very low value of the loss. In [14], the Anti-Backdoor Learning (ABL) defense uses the loss to isolate the poisoned instances during the training and then unlearns the poisoned data through gradient ascent.

3 Variance-based Defense

3.1 Threat Model

In this paper, we assume that the attacker has full access to the training database. The attacker aims to modify the model's behavior so that any input patched with her attack pattern is classified as her target class, while maintaining good categorization performance on pristine inputs to remain undetected. To implement this attack, she employs a combination of BadNets [7] and Blended [3] attacks. Given an attack pattern/trigger $\mathbf{p}$, its corresponding binary mask $\mathbf{m}$, the target label $y_t \in \mathcal{Y}$, and a blending factor $\alpha \in [0, 1]$, she selects a subset of the training dataset with a ratio r and generates malicious data according to Equation 1, before adding it to the training set with her target label. The poisoned dataset is then distributed or made available to download. Training a model on this poisoned dataset will cause the model to capture a relationship between the attack trigger $\mathbf{p}$ and the target label y_t.

$$\tilde{\mathbf{x}} = \Gamma(\mathbf{x}, \mathbf{p}, \mathbf{m}, \alpha) = \mathbf{x} \odot (\mathbf{1} - \mathbf{m}) + (1 - \alpha) \cdot (\mathbf{x} \odot \mathbf{m}) + \alpha \cdot (\mathbf{m} \odot \mathbf{p}) \quad (1)$$

where $\mathbf{x} \in \mathcal{X}$, with $\mathcal{X} \subseteq \mathbb{R}^{H \times W \times C}$, is an input image of height H, width W and color dimension C, $\odot$ the element-wise tensor product operator and $\mathbf{1}$ the all-ones tensor of the same dimension as m.

3.2 Motivation

Before using a dataset to train a model, it is important to sanitize it and ensure that no malicious data leading to a blended or a BadNets backdoor are present. To achieve this, we propose an algorithm to detect and extract the trigger pattern from the training dataset. Using the extracted patterns, the dataset can be sanitized by removing the data containing these patterns.

We want to extract the pattern $\mathbf{p_t}$ and its binary mask $\mathbf{m_t}$ for a target label y_t based on the following hypotheses:

1. when a pristine input is patched with the pattern $\mathbf{p_t}$, the model must predict y_t, which means that $\mathbf{p_t}$ and $\mathbf{m_t}$ should minimize the training loss for the set of clean data ($\mathcal{D}_C$) patched with $\mathbf{p_t}$ and associated with the target label y_t,
2. the pattern should also be as small as possible, i.e., only a few elements of the binary mask should be set to 1 (this is equivalent to using a L_0 norm penalization on $\mathbf{p_t}$) in order to keep the malicious data unnoticeable among the training instances,
3. the pixels of the pattern should be located at low-variance positions in a variance image computed from a set of malicious data with the label y_t ($\mathcal{D}_{P_t}$), since we are defending against a static-trigger data poisoning.

Given these three hypotheses, we can formulate the following minimization problem to compute $(\mathbf{p_t}, \mathbf{m_t})$:

$$\min_{p,m} \frac{1}{|\mathcal{D}_C|} \sum_{(\mathbf{x}^{(k)}, y^{(k)}) \in \mathcal{D}_C} \ell(f(\Gamma(\mathbf{x}^{(k)}, \mathbf{p}, \mathbf{m}, \alpha)), y_t) + \lambda \cdot \|\mathbf{m}\|_1 + \gamma \cdot \|\mathbf{m} \odot V_{\mathcal{D}_{P_t}}\|_1 \quad (2)$$

where f is the poisoned model learned from the malicious dataset, $\mathcal{D}_C$ the clean part of the training set, $\mathcal{D}_{P_t}$ the poisoned part with label y_t, ℓ the loss function, usually the Cross-Entropy loss, λ and γ the loss terms weighting factors and $V_{\mathcal{D}_{P_t}}$ the mean, over the color channel, of the min-max-scaled variance of the data in $\mathcal{D}_{P_t}$ such that:

$$[V_{\mathcal{D}_{P_t}}]_{i,j} = \frac{1}{n_r} \sum_{c \in \{1,\dots,n_r\}} \frac{\text{var}(X_{i,j,c}; \mathcal{D}_{P_t}) - \min_{k,l}(\text{var}(X_{k,l,c}; \mathcal{D}_{P_t}))}{\max_{k,l}(\text{var}(X_{k,l,c}; \mathcal{D}_{P_t})) - \min_{k,l}(\text{var}(X_{k,l,c}; \mathcal{D}_{P_t}))} \quad (3)$$

with $X_{i,j,c}$ the random variable associated with the pixel/variable $x_{i,j,c}$ where c is the color channel index, n_r the channel dimension and the variance given by:

$$\text{var}(X_{k,l,c}; \mathcal{D}_{P_t}) = \frac{1}{|\mathcal{D}_{P_t}|} \sum_{(\mathbf{x}^{(n)}, y^{(n)}) \in \mathcal{D}_{P_t}} (\mathbf{x}^{(n)}_{k,l,c} - \overline{X_{k,l,c}})^2 \quad (4)$$

Solving the problem 2 may be time-consuming and difficult since $\mathcal{D}_C$ and $\mathcal{D}_{P_t}$ are not known, and it may even lead to adversarial noises, especially if the model is not robust.
Instead of solving directly this minimization problem, we propose in the next section a heuristic to approximate the patterns.

3.3 Method

The proposed data sanitization method is described in Algorithm 1 and illustrated step by step in Figure 1. It consists of seven main steps, listed as follows:

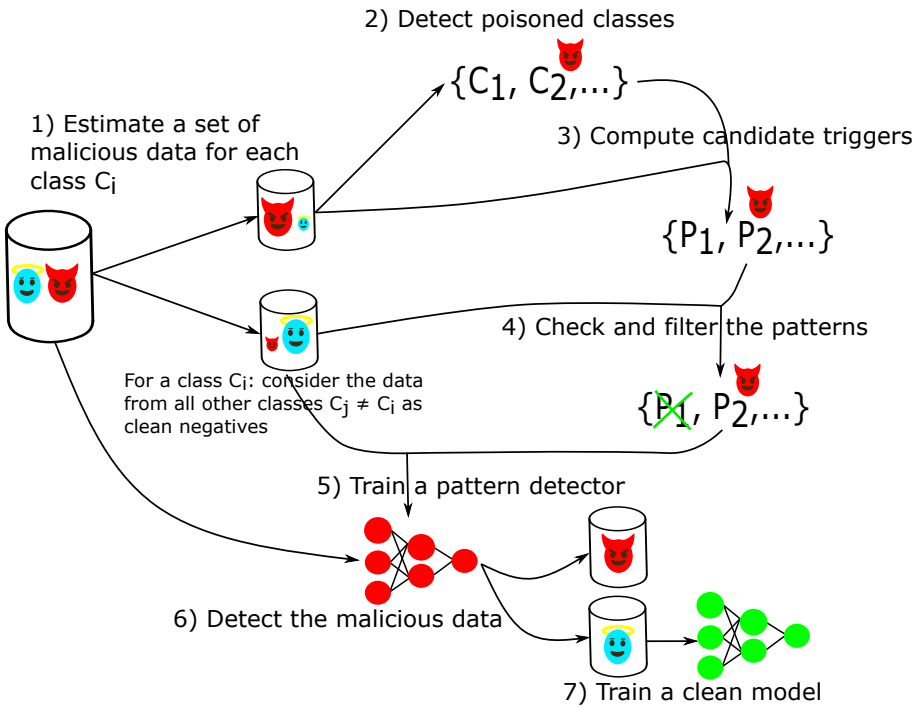


Fig. 1: Workflow of the proposed sanitization method.

1. Estimate a set of malicious instances $\tilde{\mathcal{D}}_P$
2. Detect the potentially poisoned classes
3. Compute a candidate pattern for each poisoned class
4. Check the patterns on an estimated $\tilde{\mathcal{D}}_C$
5. Train a pattern detector for each detected pattern
6. Detect the malicious instances
7. Train a clean model

In the following paragraphs, we provide a detailed description of each step.

Stage 1: Malicious instance set estimation We assume that the patterns used to poison the training set are small and simple, such that a simple classifier with a single convolution layer can memorize the relation between the pattern and the target label. The idea is to train a simple convolution model with a few layers as the one shown in Figure 2 on the training set to overfit the attack pattern. The model is thus expected to perform well on the poisoned instances and have a high generalization error on the clean data. We refer to this model as the *simple poisoned model*, denoted f_s. A subset of the poisoned data, for a class c_i, can then be estimated by selecting the top N instances with the label c_i and correctly predicted with the highest probability score (N being a parameter, knowing that for our experiments we set $N = 20$).

Algorithm 1: Variance-based defense

Data: $\mathcal{D}$: the training set to sanitize
Result: $\mathcal{D}_P$: the set of malicious instances and f_c: the sanitized model
foreach *class* $c_i \in \mathcal{Y}$ **do**
  $\tilde{\mathcal{D}_{P_i}} \leftarrow$ Estimate a set of malicious instances with label c_i ;
  if c_i *is detected as poisoned* **then**
    $(\mathbf{p_i}, \mathbf{m_i}) \leftarrow$ Compute a candidate pattern and its mask based on $\tilde{\mathcal{D}_{P_i}}$;
  end
end
$\mathcal{P} \leftarrow$ Check and filter the candidate pattern set $\{(\mathbf{p_i}, \mathbf{m_i})\}_i$;
foreach $(\mathbf{p_i}, \mathbf{m_i}) \in \mathcal{P}$ **do**
  $h_i \leftarrow$ Train a model to detect the instances patched with $\mathbf{p_i}$;
  $\mathcal{D}_{P_i} \leftarrow$ Use h_i to identify the malicious instances in $\mathcal{D}$;
end
$\mathcal{D}_P \leftarrow \bigcup_i \mathcal{D}_{P_i}$;
$f_c \leftarrow$ Train a clean model using $\mathcal{D}$ and $\mathcal{D}_P$;

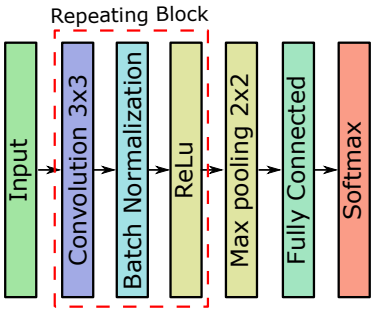


Fig. 2: Architecture of the classifier used to overfit the attack patterns.

Stage 2: Poisoned classes detection At this stage, we aim to filter out the classes considered as non-poisoned, retaining only the suspicious ones for further analysis. To achieve this, for each class c_i we identify the most important (pixel) variable that is commonly used by the poisoned model f_s to correctly classify the instances of the estimated malicious set $\tilde{\mathcal{D}_{P_i}}$. If the empirical distributions of the importance of this variable in $\tilde{\mathcal{D}_{P_i}}$ and in a subset of potential clean-instances from c_i are the same then we can assume that the class c_i is not poisoned. To find the most important variable, given the *simple poisoned model* f_s, we compute the gradient of the first loss term of Equation 2 w.r.t. the malicious input, as given by Equation 5.

$$\nabla L(f_s, \tilde{\mathcal{D}_{P_i}}) = \frac{1}{|\tilde{\mathcal{D}_{P_i}}|} \sum_{(\mathbf{x}^{(k)}, c_i) \in \tilde{\mathcal{D}_{P_i}}} \frac{\partial \ell}{\partial \mathbf{x}}(f_s(\mathbf{x}^{(k)}), c_i) \tag{5}$$

As we want the most important pixels explaining the decision, we choose the following loss function $\ell(\mathbf{x}, c)$:

$$\ell(\mathbf{x}, c) = P(c|\mathbf{x}; f_s) - \max_{y \in \mathcal{Y} - \{c\}} P(y|\mathbf{x}; f_s) \tag{6}$$

$P(c|\mathbf{x}; f_s)$ is given by the c^{th} component of the softmax layer of f_s and in order to avoid numerical instabilities, we use the logit layer instead of the softmax layer.

For poisoned instances, some components of the gradient (Eq. 5) will have a large absolute value compared to the clean-instances. Indeed, the partial derivative at given coordinates indicates the sensitivity of the loss function to changes in that pixel's value. In the case of poisoned instances, the backdoor pattern is designed to strongly influence the model's prediction. As a result, only a few pixels, those containing the trigger, tend to dominate the gradient signal. We thus assume that, for poisoned instances, a small number of pixels will have a large impact on the prediction. The most influential pixel can therefore be identified by locating the index (i^*, j^*) corresponding to the maximum absolute value of the mean gradient across the color channels:

$$i^*, j^* = \arg\max_{i,j} |\text{mean}_c \left[\nabla L(f_s, \tilde{\mathcal{D}_{P_i}})\right]_{i,j,c}| \tag{7}$$

We now define the importance of this pixel in the decision of an input $\mathbf{x}$ by f_s, as follows:

$$I(\mathbf{x}, c_i; f_s, i^*, j^*) = \max_c [|\nabla L(f_s, \{(\mathbf{x}, c_i)\})|]_{i^*, j^*, c} \tag{8}$$

The two-sided Kolmogorov–Smirnov test [5] is used with a confidence level of 99% to compare the distributions of $I(\mathbf{x}, c_i; f_s, i^*, j^*)$ between $\tilde{\mathcal{D}_{P_i}}$ and a set of potential clean-instances from c_i. We use the instances of c_i with the lowest prediction scores as clean-instances. When the p-value is below 0.01, we consider c_i as potentially poisoned.

Stage 3: Candidate pattern computation With the *simple poisoned model* f_s and the estimated subset of malicious instances of the class c_i, we can compute an approximate pattern by using the gradient defined in Equation 5 which gives the importance of the pixels w.r.t. the loss function. To obtain a binary mask from the gradient, the following processing is performed:

1. Flattening the gradient of Eq. 5 by taking the L_2 norm of the gradient over the color channel
2. Min-Max-scaling the result to have the values within $[0, 1]$
3. Binarizing the values using a threshold

The binarization step may be tricky as it involves defining a threshold which may depends on the input data. To avoid manually defining this threshold, we

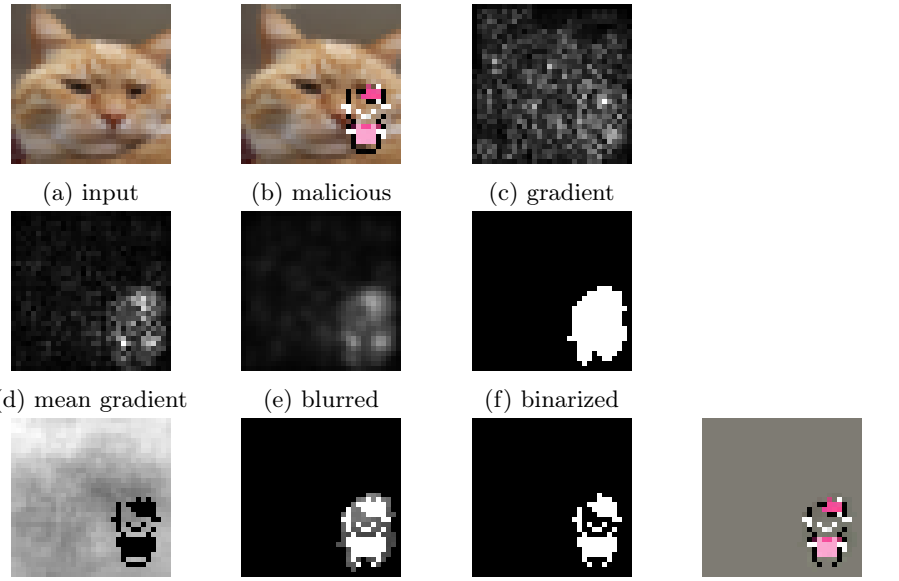
used the Otsu method, which finds the best threshold that minimizes the intra-variance [20]. In order to capture also the neighboring points of an important point, we apply a Gaussian blur before applying the Otsu method.
Once we have the binary gradient mask, we need to select only the points with the lowest variance, which requires a variance threshold. The variance matrix on $\tilde{\mathcal{D}_{P_i}}$ can be computed with the Equation 3. The final candidate mask is obtained by performing an element-wise product between the gradient mask and the variance matrix before binarizing with the Otsu method (note that to keep the points with low variances, we use 1 minus the variance matrix).
To have the candidate pattern, we apply the candidate mask on the mean image of $\tilde{\mathcal{D}_{P_i}}$. Algorithm 2 describes the whole process of generating the candidate pattern and mask.

Algorithm 2: Pattern Computation

Data: $\tilde{\mathcal{D}_{P_i}}$: a set of malicious instances for the class c_i, f_s: the poisoned model
Result: $(\mathbf{p_i}, \mathbf{m_i})$): the candidate pattern and mask for the class c_i
$\mathbf{g} \leftarrow$ compute the gradient with Equation 5 using the logit layer ;
foreach *pixel at coordinate* i, j **do**
  $\tilde{g}_{i,j} \leftarrow \sqrt{\sum_r g_{i,j,r}^2}$;
end
$\mathbf{g}_{scaled} \leftarrow \text{min\_max\_scale}(\tilde{\mathbf{g}})$;
$\mathbf{g}_{blured} \leftarrow \text{gaussian\_blur}(\mathbf{g}_{scaled})$;
$\mathbf{g}_m \leftarrow \text{otsu\_binarization}(\mathbf{g}_{blured})$;
$\mathbf{V}_{\tilde{\mathcal{D}_{P_i}}} \leftarrow$ Use Equation 3 to compute the variance on $\tilde{\mathcal{D}_{P_i}}$;
$\mathbf{m_i} \leftarrow \text{otsu\_binarization}(\mathbf{g}_m \odot (\mathbf{1} - V_{\tilde{\mathcal{D}_{P_i}}}))$;
$\mathbf{p_i} \leftarrow \mathbf{m_i} \odot \frac{1}{|\tilde{\mathcal{D}_{P_i}}|} \sum_{(\mathbf{x}^{(k)}, y^{(k)}) \in \tilde{\mathcal{D}_{P_i}}} \mathbf{x}^{(k)}$;

Stage 4: Check and filter the patterns At this stage, a candidate pattern and its mask have been calculated for each potentially poisoned class. We need to check that it triggers a malicious behavior of the *simple poisoned model*. To this end, we patch the pattern to instances from other classes (estimated $\tilde{\mathcal{D}_C}$) before feeding them to the model, and we check whether the model assigns them to the class associated with the pattern. We compute the Attack Success Rate (ASR), i.e., the proportion of patched instances predicted as the attack label. If this rate falls below a user-defined threshold, we remove the pattern from the candidate set. The remaining patterns are then considered harmful.

Stage 5: Train a pattern detector To detect the presence of a pattern in the training set, we propose to train a binary classifier for each pattern. We use a one-layer-convolution network as shown in Figure 2 except that 1) we remove



(a) input (b) malicious (c) gradient

(d) mean gradient (e) blurred (f) binarized

(g) Variance (h) grad⊙(1-var) (i) re-binarized (j) Extracted pattern

Fig. 3: Illustration of the candidate pattern computation process. The first row shows (a) an example of input image from the cat class, (b) its malicious version labeled with the attacker's label and (c) its gradient (Eq. 5). In the last two rows, the pattern computation process, described in the Algorithm 2, is illustrated step by step. The gray background in (j) represents transparency.

the batch normalization, 2) we add a dropout layer before the fully connected layer in order to reduce the overfitting and 3) we replace the softmax layer by a sigmoid layer. For a given pattern $\mathbf{p_i}$ for the class c_i, we build the training data as follows, we use the instances of the other classes and label them as "0" (clean), and we patch these instances using the Equation 1 with random blending factors, before labeling them as "1" (poisoned). Instead of training directly on the images, we can achieve better performance by training the model on the image gradient using Equation 5 on a single image. Moreover, we use Semi-Supervised Learning (SSL) with Pseudo-labels [13]. After a few epochs of supervised learning, we use the classifier to label the data of the class c_i and we add these data to the training set. We continue the SSL by relabeling the data after each epoch for a predefined number of epochs to make the pseudo-labels stable. The Algorithm 3 describes the training procedure.

Stage 6: Detect the malicious instances We compute the gradient for each instance of the training set, and we feed it to each trained pattern detector. We label the instance as malicious if at least one pattern detector labeled it as "1".

Algorithm 3: Training a Pattern Detector

Data: $\mathcal{D}$: the (poisoned) training set, f_s the poisoned model learned in the first stage, $(\mathbf{p_i}, \mathbf{m_i}, c_i)$ the attack pattern $\mathbf{p_i}$ with its mask $\mathbf{m_i}$ for the target class c_i and h_i the binary classifier to fit
Result: h_i: the trained pattern detector for $(\mathbf{p_i}, \mathbf{m_i}, c_i)$
$\mathcal{D}_T \leftarrow \mathcal{D} - \{(\mathbf{x}, y) \in \mathcal{D} : y \neq c_i\}$;
$\mathcal{D}_{ssl} \leftarrow \{\}$;
$(\mathcal{D}_t, \mathcal{D}_v, \mathcal{D}_e) \leftarrow$ split $\mathcal{D}_\mathcal{T}$ into training, validation, and test sets;
for *epoch* $\leftarrow 0$ **to** *max_ epoch* **do**
  foreach *batch* $\mathcal{B}$ *of* $\mathcal{D}_t$ **do**
    $\mathcal{U} \leftarrow \{\}$;
    foreach $(x, y) \in \mathcal{B}$ **do**
      $\alpha \leftarrow$ random$(0.1, 1)$;
      $\mathbf{z} \leftarrow \Gamma(\mathbf{x}, \mathbf{p_i}, \mathbf{m_i}, \alpha)$ (Eq. 1);
      $\mathcal{U} \leftarrow \mathcal{U} \cup \{(\nabla L(f_s, \{(\mathbf{x}, c_i)\}), 0), (\nabla L(f_s, \{(\mathbf{z}, c_i)\}), 1)\}$ (Eq. 5) ;
    end
    Optimize h_i with $\mathcal{U} \cup \mathcal{D}_{ssl}$;
  end
  if *start_ ssl_ epoch* $<$ *epoch* $<$ *stop_ relabeling_ epoch* **then**
    Use h_i to label $\{(\mathbf{x}, y) \in \mathcal{D} : y = c_i\}$;
    $\mathcal{D}_{ssl} \leftarrow$ the pseudo-labeled data ;
  end
end

Stage 7: Train a clean model Once we have identified the malicious instances, we can train a clean model from scratch or fine-tune a poisoned model by using both the clean instance set and the poisoned one.

Robustness The proposed method relies on a *simple poisoned model*, which is assumed to separate poisoned data from clean data based on the prediction score. This implies that the model must be complex enough to capture the trigger (high attack success rate) and simple enough to underperform on clean data (low accuracy rate). Relying on a single model to achieve this task might not be robust. To address this issue, we propose to use an ensemble of simple models based on the architecture shown in Figure 2 with, e.g., different number of filters and layers. We use each model of the ensemble, simultaneously and independently, to perform Stage 1 to Stage 4. At the end of Stage 4, for each model, we have extracted a set of triggers, one for each detected poisoned class. For robustness, a class is considered poisoned only if the number of extracted patterns for that class exceeds a certain threshold, e.g., half the number of models in the ensemble (majority vote). However, in this work, we adopt a more defensive approach to reduce the false negative rate. Therefore, we consider a class as poisoned if at least one trigger has been extracted for this class. To perform the Stages 5 and 6, only one model and pattern must be retained for a given poisoned class. To select the most appropriate model-and-pattern pair for a class, we choose the

one with the lowest accuracy on the training set (computed in Stage 1 during the training) and the highest ASR (computed in stage 4), i.e., the pair that achieves the highest score according to the following metric: $\alpha \cdot (1 - \text{acc}) + (1 - \alpha) \cdot \text{asr}$. In our work, we set $\alpha = 0.6$ to slightly favor the model with the lowest accuracy.

4 Experiments

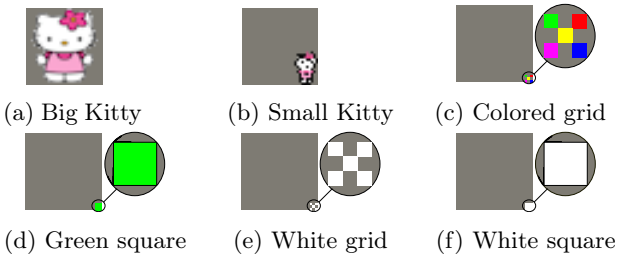
4.1 Experimental Setup

For the evaluation, we used the BackdoorBench framework [26], and our code is available online[4]. We compared our method, *VBD*, against three high-performing state-of-the-art methods [28]: Anti-Backdoor Learning (ABL) [14], AGPD [28], and SCAn [25]. For VBD, we used an ensemble of 3 models: a 1-layer model with 64 filters, a 2-layers model with 10 filters each, and a 2-layers model with 16 filters each, following the architecture shown in Figure 2. We used 80% of the training set only to train the models and the remaining 20% for the validation, detection, pattern extraction, and detector training. For training the poisoned and detector models, we used a learning rate of 0.01, a batch size of 256 for CIFAR-10 and 64 for Tiny ImageNet, and 20 training epochs. For the competitors, we used the default settings, which include the use of the Pre-Act ResNet18 model [9], knowing that AGPD and SCAn require also an additional dataset of clean images, which, by default, consists of 10 images per class taken from the test set. Full implementation details and additional parameters are available in the released code repository.
The experiments consist of poisoning 10% of the training dataset using the BadNets and Blended attacks and evaluating the performance of the four methods in detecting the poisons in the training set. For the Blended attacks, we evaluated 3 blending factors: 10%, 20% and 50%. A blending factor of 10% means that the trigger is 90% transparent. A BadNets attack is equivalent to a Blended attack with a blending factor of 100% (full opacity). We used both an All-to-One attack, where only one class is poisoned with 10% of the images from each class (including the target class) and an All-to-All attack, where all classes are poisoned such that class $(k + 1 \bmod N_c)$, with N_c being the number of classes in the dataset, is poisoned with 10% of the images from class k.

For the database, we used both CIFAR-10 [11] and Tiny ImageNet [12]. CIFAR-10 is composed of 10 classes with 5000 training images of size 32×32 per class. Tiny ImageNet contains 200 classes with 500 training images of size 64×64 per class. For AGPD and SCAn, which require 10 auxiliary clean images per class in their default settings, we used a total of 100 and 2000 clean images from the test set for CIFAR-10 and Tiny ImageNet, respectively.

We used six attack triggers to evaluate the methods, as shown in Figure 4. When more than one class is poisoned (All-to-All), we used only the grid and square triggers, positioning them at different locations based on the class index. The placement starts at the bottom right of the image and moves from right to

[4] `https://github.com/Orange-OpenSource/BackdoorBench/tree/vbd-v1`



(a) Big Kitty (b) Small Kitty (c) Colored grid

(d) Green square (e) White grid (f) White square

Fig. 4: Attack patterns used in the evaluation. All the triggers, except for the two "kitty" patterns, are of size 3×3. The "Big Kitty" spans the entire image and the "Small Kitty" has a size of 10×14. The gray background is transparent.

left and bottom to top to avoid overlapping. We ran a total of 80 experiments: for both CIFAR-10 and Tiny ImageNet, there were 24 experiments (6 triggers $\times$ 4 blending factors) for the All-to-One setting and 16 experiments (4 triggers $\times$ 4 blending factors) for the All-to-All case. Each experiment is repeated 5 times. In each repetition $i \in \{0, \ldots, 4\}$, the random seed is set to i, and for the All-to-One attacks, the target class is also set to i. The F_1-scores and the averaged F_1-scores are reported as the mean over the five repetitions, along with their standard deviation.

4.2 Evaluation Metrics

To evaluate the performance of poisoned instance detection, we use the F_1 score, defined as follows:

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}, \quad \text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \quad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

where TP, TN, FP, and FN denote the number of true positives, true negatives, false positives, and false negatives, respectively.
Precision measures the proportion of correctly identified poisoned instances among all instances classified as poisoned. Recall measures the proportion of actual poisoned instances that were correctly identified by the classifier. Since it is important for a method to perform well on both metrics, the F_1 score, defined as the harmonic mean of precision and recall, is used to provide a balanced evaluation.

4.3 Experimental Results

Table 1 provides the F_1-score of the defense methods for the All-to-One BadNets attack. SCAn and VBD achieve the best results. On average, SCAn outperforms VBD by 0.34% on CIFAR-10, while VBD performs better on Tiny ImageNet, with an improvement of 1.59%. The average F_1-score across the 12 experiments indicates that VBD surpasses SCAn by 0.62%. Notably, AGPD exhibits low

detection performance on the "Big Kitty" pattern, as this pattern covers the entire image, making it difficult for AGPD to distinguish between poisoned and clean instances.

Table 1: All-to-One BadNets poisoned-instance detection F_1-score (%).

Set	Pattern	VBD	ABL	AGPD	SCAn
CIFAR-10	big kitty	99.87 ±00.20	55.39 ±23.40	79.35 ±44.37	**100.00 ±00.00**
	small kitty	99.90 ±00.14	82.12 ±10.54	99.62 ±00.28	**100.00 ±00.00**
	color grid	99.58 ±00.15	90.21 ±03.01	91.36 ±12.88	**100.00 ±00.00**
	green square	99.34 ±00.25	84.04 ±12.19	90.93 ±11.69	**99.99 ±00.01**
	white grid	99.65 ±00.09	89.66 ±02.50	94.60 ±02.64	**100.00 ±00.00**
	white square	97.46 ±00.31	84.30 ±02.13	93.66 ±02.96	**97.86 ±00.12**
	Average	99.30 ±00.10	80.95 ±04.01	91.59 ±11.65	**99.64 ±00.02**
Tiny ImageNet	big kitty	**99.98 ±00.02**	81.69 ±08.87	00.00 ±00.00	94.41 ±12.51
	small kitty	99.98 ±00.02	94.58 ±07.34	88.90 ±24.04	**100.00 ±00.00**
	color grid	99.86 ±00.12	96.40 ±02.82	98.69 ±00.19	**99.95 ±00.04**
	green square	**99.72 ±00.39**	96.77 ±01.67	98.21 ±00.66	95.34 ±10.35
	white grid	**98.70 ±02.64**	95.93 ±00.84	98.34 ±00.73	97.39 ±05.63
	white square	95.89 ±02.35	88.66 ±06.24	59.19 ±54.03	**97.50 ±01.26**
	Average	**99.02 ±00.68**	92.34 ±02.61	73.89 ±08.12	97.43 ±03.16
Average		**99.16 ±00.32**	86.65 ±00.85	82.74 ±08.96	98.54 ±01.58

For the All-to-All BadNets attacks, SCAn fails on both CIFAR-10 and Tiny ImageNet as shown in Table 2. It appears that when at least half of the classes are poisoned SCAn is unable to detect the attack. On the contrary, VBD achieves the best F_1-performance, outperforming AGPD by 8.69% on CIFAR-10 and ABL by 23% on Tiny-ImageNet.

Table 2: All-to-All BadNets poisoned-instance detection F_1-score (%).

Set	Pattern	VBD	ABL	AGPD	SCAn
CIFAR-10	color grid	**94.37 ±01.03**	56.76 ±12.58	68.28 ±14.05	00.00 ±00.00
	green square	**91.00 ±02.00**	55.36 ±12.30	86.66 ±15.02	00.00 ±00.00
	white grid	**95.64 ±03.54**	58.06 ±07.48	93.07 ±09.05	00.00 ±00.00
	white square	**93.51 ±00.82**	53.14 ±12.94	91.73 ±08.45	00.00 ±00.00
	Average	**93.63 ±00.66**	55.83 ±07.63	84.94 ±07.88	00.00 ±00.00
Tiny ImageNet	color grid	**83.91 ±02.75**	50.57 ±03.50	33.30 ±00.63	00.00 ±00.00
	green square	**85.00 ±01.24**	38.47 ±08.08	34.14 ±01.30	00.00 ±00.00
	white grid	**52.27 ±02.68**	43.45 ±06.51	34.34 ±00.97	00.00 ±00.00
	white square	**44.82 ±02.46**	41.36 ±08.17	33.84 ±00.67	00.00 ±00.00
	Average	**66.50 ±01.91**	43.46 ±02.41	33.90 ±00.54	00.00 ±00.00
Average		**80.06 ±01.14**	49.65 ±04.32	59.42 ±04.14	00.00 ±00.00

The performance results for the Blended attacks in both All-to-One and All-to-All settings are provided in Table 3 and Table 4, respectively. As we can see, in the All-to-One setting, the results are comparable to those in the All-to-One BadNets case, with SCAn and VBD leading. Nevertheless, in this case, VBD outperforms SCAn by 0.24% on CIFAR-10 and by 7.37% on Tiny ImageNet. The averaged F_1 score across the experiments on both datasets shows that VBD outperforms SCAn by 3.8%. For the All-to-All Blended attack case, as previously mentioned, SCAn fails, and AGPD outperforms the other defenses, with VBD finishing second. It is noteworthy that VBD struggles to detect the attack and extract the triggers when the blending factor is below 50%. However, when the blending factor is 50%, VBD performs better than AGPD.

Table 3: All-to-One Blended poisoned-instance detection mean F_1-score (%) over the 6 triggers shown in Figure 4.

Set	Blended	VBD	ABL	AGPD	SCAn
CIFAR-10	10%	**93.34 ±01.08**	68.60 ±04.12	90.38 ±07.27	92.80 ±06.14
	20%	**96.85 ±00.75**	76.58 ±01.18	90.22 ±08.70	96.19 ±01.10
	50%	98.68 ±00.54	84.40 ±01.90	90.68 ±09.68	**99.14 ±00.10**
	Average	**96.29 ±00.72**	76.53 ±01.99	90.43 ±05.73	96.05 ±02.01
Tiny ImageNet	10%	**83.12 ±06.42**	78.03 ±03.02	65.02 ±09.46	75.80 ±03.19
	20%	**93.62 ±05.91**	87.03 ±02.73	72.76 ±19.37	84.58 ±02.84
	50%	**98.05 ±00.99**	92.01 ±01.67	79.50 ±13.01	92.31 ±03.99
	Average	**91.60 ±03.80**	85.69 ±01.33	72.43 ±11.90	84.23 ±02.28
Average		**93.94 ±01.83**	81.11 ±01.31	81.43 ±07.04	90.14 ±02.07

Table 4: All-to-All Blended poisoned-instance detection mean F_1-score (%) over the 4 grid and square triggers shown in Figure 4.

Set	Blended	VBD	ABL	AGPD	SCAn
CIFAR-10	10%	56.16 ±03.38	14.08 ±06.64	**70.70 ±16.82**	00.00 ±00.00
	20%	72.08 ±01.54	43.97 ±05.78	**75.29 ±12.39**	00.00 ±00.00
	50%	**91.68 ±01.27**	53.31 ±06.28	84.47 ±04.86	01.40 ±03.13
	Average	73.31 ±00.80	37.12 ±05.43	**76.82 ±05.14**	00.47 ±01.04
Tiny ImageNet	10%	00.00 ±00.00	02.84 ±00.12	**20.81 ±08.53**	00.00 ±00.00
	20%	09.15 ±04.93	08.80 ±03.78	**29.04 ±07.64**	00.00 ±00.00
	50%	**42.55 ±01.09**	37.17 ±03.07	35.05 ±00.18	00.00 ±00.00
	Average	17.23 ±01.69	16.27 ±01.94	**28.30 ±05.07**	00.00 ±00.00
Average		45.27 ±00.76	26.69 ±02.38	**52.56 ±02.43**	00.23 ±00.52

Table 5 provides a summary of the performance of the tested defense methods, with the F_1-score averaged over all the 80 experiments we conducted. VBD

ranks first on CIFAR-10 and, notably, on Tiny ImageNet. It outperforms AGPD by 3.61% on CIFAR-10, and ABL by 6.25% on Tiny ImageNet. SCAn is penalized by its failure in the All-to-All settings. It is worth noting that, unlike AGPD and SCAn, VBD and ABL do not rely on an auxiliary dataset. The standard deviation also shows that VBD is a stable method.

Table 5: Summary of the poisoned-instance detection F_1-score (%) averaged over all the 80 experiments.

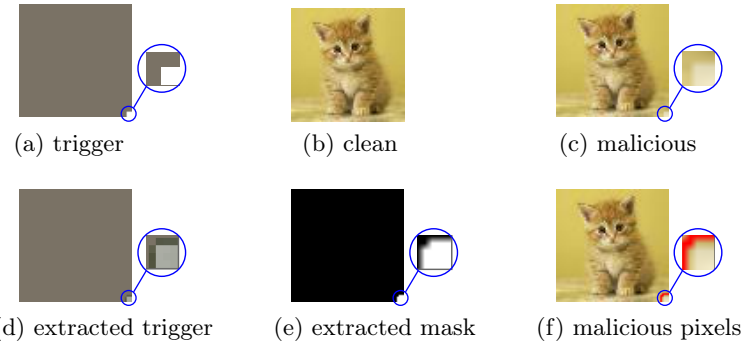
Set	VBD	ABL	AGPD	SCAn
CIFAR-10	**89.58 ±00.54**	63.30 ±03.02	85.97 ±03.60	58.31 ±00.60
Tiny ImageNet	**67.89 ±01.64**	61.64 ±00.94	55.56 ±06.94	52.52 ±01.36
All	**78.74 ±00.77**	62.47 ±01.28	70.76 ±04.75	55.41 ±00.96

4.4 Discussion on Explainability

The proposed method extracts the salient part of the attack trigger along with its binary mask. The extracted trigger provides a quick and general visual explanation of the attack. Moreover, the binary mask can be used to isolate the pixels responsible for the malicious behavior of an instance detected as poisoned. Figure 5 illustrates the explainability of the method on a blended attack on Tiny ImageNet, using a 3×3 white square trigger (shown in subfigure 5a) with a blending factor of 50%. The VBD method computes both the trigger and its corresponding mask, shown in subfigures 5d and 5e, respectively. The extracted pattern approximates the malicious trigger. Although the method cannot fully recover the original colors of the trigger due to the blending, the result remains sufficiently expressive to identify and explain the attack. The generated mask can then be applied to any instance detected as poisoned to isolate the malicious pixels and explain the VBD decision. The instance in subfigure 5c is detected by VBD as poisoned. The extracted mask in subfigure 5e is then used to localize the malicious pixels, as shown in subfigure 5f, where these pixels are outlined in red.

5 Conclusion

In this article, we proposed an efficient defense algorithm against BadNets and Blended attacks. Our method does not require additional information, such as a clean dataset, and can be directly applied to a training set to detect poisoned instances. A key advantage of our approach is its explainability: it extracts the harmful part of the attack trigger, enabling experts to better understand the nature of the attack. Moreover, our defense is effective in both the All-to-One setting, where only one class is poisoned, and the more challenging All-to-All



(a) trigger (b) clean (c) malicious

(d) extracted trigger (e) extracted mask (f) malicious pixels

Fig. 5: Illustration of the method's explainability on a blended attack using a white square trigger (a) with a blending factor of 0.5. The pixels responsible for the malicious misclassification are outlined in red in (f). In (a) and (d), transparency is represented by the gray background.

setting, where all classes are affected. Experimental evaluations on two well-known image datasets, compared against three state-of-the-art defense methods, demonstrate the strong performance of our approach.

Our current method focuses on static-trigger attacks, which represent the simplest and most realistic backdoor attack scenarios, as they require minimal prerequisites and are thus more likely to be deployed by attackers in real-world settings. However, our approach can be extended to dynamic triggers, where the trigger is not fixed but can appear in predefined locations [23]. This could be achieved, for instance, by clustering the estimated poisoned set based on variance before extracting the trigger pattern.

In future work, we plan to further evaluate this extension and explore defenses against a broader range of backdoor attack strategies.

References

1. Chen, B., Carvalho, W., Baracaldo, N., Ludwig, H., Edwards, B., Lee, T., Molloy, I.M., Srivastava, B.: Detecting backdoor attacks on deep neural networks by activation clustering. In: SafeAI@AAAI. CEUR Workshop Proceedings (2019)
2. Chen, X., Liu, C., Li, B., Lu, K., Song, D.: Targeted backdoor attacks on deep learning systems using data poisoning. arXiv preprint arXiv:1712.05526 (2017)
3. Chen, X., Liu, C., Li, B., Lu, K., Song, D.: Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning. arXiv preprint arxiv:1712.05526 (2017)
4. Chou, E., Tramer, F., Pellegrino, G.: Sentinet: Detecting localized universal attacks against deep learning systems. In: IEEE SPW (2020)
5. Dodge, Y.: Kolmogorov–smirnov test. In: The Concise Encyclopedia of Statistics. Springer New York (2008)
6. Gao, Y., Xu, C., Wang, D., Chen, S., Ranasinghe, D.C., Nepal, S.: Strip: a defence against trojan attacks on deep neural networks. In: ACSAC (2019)
7. Gu, T., Dolan-Gavitt, B., Garg, S.: BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain. arXiv preprint arxiv:1708.06733 (2017)

8. Guo, J., Li, Y., Chen, X., Guo, H., Sun, L., Liu, C.: SCALE-UP: An efficient black-box input-level backdoor detection via analyzing scaled prediction consistency. In: ICLR (2023)
9. He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. In: ECCV (2016)
10. Huang, K., Li, Y., Wu, B., Qin, Z., Ren, K.: Backdoor defense via decoupling the training process. In: ICLR (2022)
11. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Master's thesis, Department of Computer Science, University of Toronto (2009)
12. Le, Y., Yang, X.: Tiny imagenet visual recognition challenge. CS 231N (2015)
13. Lee, D.H.: Pseudo-label : The simple and efficient semi-supervised learning method for deep neural networks. In: WREPL@ICML (2013)
14. Li, Y., Lyu, X., Koren, N., Lyu, L., Li, B., Ma, X.: Anti-backdoor learning: Training clean models on poisoned data. NeurIPS **34**, 14900–14912 (2021)
15. Li, Y., Lyu, X., Ma, X., Koren, N., Lyu, L., Li, B., Jiang, Y.G.: Reconstructive neuron pruning for backdoor defense. In: ICML (2023)
16. Liu, Q., Li, P., Zhao, W., Cai, W., Yu, S., Leung, V.C.M.: A survey on security threats and defensive techniques of machine learning: A data driven view. IEEE Access (2018)
17. Ma, W., Wang, D., Sun, R., Xue, M., Wen, S., Xiang, Y.: The "beatrix" resurrections: Robust backdoor detection via gram matrices. In: NDSS (2023)
18. Nguyen, A., Tran, A.: Wanet–imperceptible warping-based backdoor attack. arXiv preprint arXiv:2102.10369 (2021)
19. Nguyen, T.A., Tran, A.: Input-aware dynamic backdoor attack. NeurIPS (2020)
20. Otsu, N.: A Threshold Selection Method from Gray-Level Histograms. IEEE Transactions on Systems, Man, and Cybernetics (1979)
21. Pitropakis, N., Panaousis, E., Giannetsos, T., Anastasiadis, E., Loukas, G.: A taxonomy and survey of attacks against machine learning. Computer Science Review **34** (2019)
22. Qi, X., Xie, T., Li, Y., Mahloujifar, S., Mittal, P.: Revisiting the assumption of latent separability for backdoor defenses. In: ICLR (2023)
23. Salem, A., Wen, R., Backes, M., Ma, S., Zhang, Y.: Dynamic backdoor attacks against machine learning models. In: EuroS&P (2022)
24. Selvaraju, R.R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., Batra, D.: Grad-cam: visual explanations from deep networks via gradient-based localization. IJCV (2020)
25. Tang, D., Wang, X., Tang, H., Zhang, K.: Demon in the variant: Statistical analysis of {DNNs} for robust backdoor contamination detection. In: USENIX Security Symposium (2021)
26. Wu, B., Chen, H., Zhang, M., Zhu, Z., Wei, S., Yuan, D., Shen, C.: Backdoorbench: A comprehensive benchmark of backdoor learning. In: NeurIPS (2022)
27. Wu, D., Wang, Y.: Adversarial neuron pruning purifies backdoored deep models. In: NeurIPS (2021)
28. Yuan, D., Wei, S., Zhang, M., Liu, L., Wu, B.: Activation gradient based poisoned sample detection against backdoor attacks. arXiv preprint arxiv:2312.06230 (2024)
29. Zeng, Y., Park, W., Mao, Z.M., Jia, R.: Rethinking the backdoor attacks' triggers: A frequency perspective. In: IEEE ICCV (2021)
30. Zhang, M., Zhu, M., Zhu, Z., Wu, B.: Reliable poisoned sample detection against backdoor attacks enhanced by sharpness aware minimization. arXiv preprint arXiv:2411.11525 (2024)