XI. Appendices

A. Proof of Lemma 1

Lemma 1. *Given l models, the probability that a trigger is effective on the k-th model is p_k^t, which is simply the percentage of a group of inputs (attached by the trigger) that are classified into the target label of the trigger. Therefore, the average effectiveness of the trigger over all the l models $p^t = \frac{1}{l}\sum_{k=1}^{l} p_k^t$. Similarly, the probability that an AP is effective on the k-th model is p_k^a, and the average effectiveness of the AP over all the l models $p^a = \frac{1}{l}\sum_{k=1}^{l} p_k^a$. For any two models in l, whether the trigger or AP is effective on one model is independent of the other model. For any small positive number ε, if $p^a - p^t > 2\varepsilon$, the number of models the AP is effective on is always larger than that of the trigger when $l \to +\infty$. Moreover, if $p^a - p^t = \lambda$, the number of models the AP is effective on is $l \cdot (\lambda - 2\varepsilon)$ larger than that of the trigger.*

Proof. Let x_k^t and x_k^a be two groups of random variables. $x_k^t = 1$ denotes that the trigger is effective on the k-th model and $x_k^t = 0$ denotes ineffective. Similarly, $x_k^a = 1$ denotes that an AP is effective on the k-th model and $x_k^a = 0$ denotes ineffective. Then, the number of models that the trigger is effective on is $\sum_{k=1}^{l} x_k^t$ and the number of models that the AP is effective on is $\sum_{k=1}^{l} x_k^a$.

$$\begin{aligned} D(x_k^t) &= E((x_k^t)^2) - E^2(x_k^t) \\ &= 0^2 \cdot (1-p_k^t) + 1^2 \cdot p_k^t - (0 \cdot (1-p_k^t) + 1 \cdot p_k^t)^2 \\ &= p_k^t - (p_k^t)^2 < 1 \end{aligned}$$

Since there exists an upper bound for $D(x_k^t)$, according to Chebyshev's law of large numbers, we have:

$$\lim_{l\to+\infty} P\{|\frac{1}{l}\sum_{k=1}^{l} x_k^t - \frac{1}{l}\sum_{k=1}^{l} E(x_k^t)| < \varepsilon\} = 1$$

$$\lim_{l\to+\infty} |\frac{1}{l}\sum_{k=1}^{l} x_k^t - \frac{1}{l}\sum_{k=1}^{l} E(x_k^t)| < \varepsilon$$

$$\lim_{l\to+\infty} |\frac{1}{l}\sum_{k=1}^{l} x_k^t - \frac{1}{l}\sum_{k=1}^{l}(0 \cdot (1-p_k^t) + 1 \cdot p_k^t)| < \varepsilon$$

$$\lim_{l\to+\infty} |\frac{1}{l}\sum_{k=1}^{l} x_k^t - p^t| < \varepsilon$$

$$\lim_{l\to+\infty} \frac{1}{l}\sum_{k=1}^{l} x_k^t - p^t < \varepsilon \tag{1}$$

Similarly, we have:

$$\lim_{l\to+\infty} |\frac{1}{l}\sum_{k=1}^{l} x_k^a - p^a| < \varepsilon$$

$$\lim_{l\to+\infty} p^a - \frac{1}{l}\sum_{k=1}^{l} x_k^a < \varepsilon \tag{2}$$

Combining (1) and (2), we have:

$$\lim_{l\to+\infty} \frac{1}{l}\sum_{k=1}^{l} x_k^t - \frac{1}{l}\sum_{k=1}^{l} x_k^a + p^a - p^t < 2\varepsilon$$

Since

$$p^a - p^t > 2\varepsilon$$

we have:

$$\lim_{l\to+\infty} \frac{1}{l}\sum_{k=1}^{l} x_k^t - \frac{1}{l}\sum_{k=1}^{l} x_k^a < 0$$

$$\lim_{l\to+\infty} \sum_{k=1}^{l} x_k^t - \sum_{k=1}^{l} x_k^a < 0$$

When

$$p^a - p^t = \lambda$$

we have:

$$\lim_{l\to+\infty} \frac{1}{l}\sum_{k=1}^{l} x_k^t - \frac{1}{l}\sum_{k=1}^{l} x_k^a - \lambda < 2\varepsilon$$

$$\lim_{l\to+\infty} \sum_{k=1}^{l} x_k^a - \sum_{k=1}^{l} x_k^t > l \cdot (\lambda - 2\varepsilon)$$

This completes the proof. □

B. Impact of Triggers' Properties

Other properties of triggers besides the size may also impact the performance of our backdoor detection approach, e.g., position, pattern, shape, connectivity (triggers of one piece or several pieces), and concurrency (multiple triggers in a model). Therefore, we also evaluate how these properties impact NeuralSanitizer by mutating one while keeping the others unchanged.

Size of Triggers. To evaluate the performance of NeuralSanitizer on the trigger size, we inject triggers of different sizes, 5%∼40% of the area of the original images into the datasets MNIST, GTSRB, CIFAR-10, YouTube-Face, and VGG-Face. Then we use NeuralSanitizer to detect the backdoors, without any pre-knowledge of the size. Overall, the results show that the average size of the triggers we can detect on the five datasets is 25%. In particular, NeuralSanitizer is able to detect triggers up to 22%, 28%, 30%, and 33% of the original images on MNIST, GTSRB, CIFAR-10, and YouTube-Face, respectively, which demonstrates its capability to detect large triggers. Regarding VGG-Face, triggers of up to 15% of the original images can be detected, compared with existing work that can only detect the triggers of around 5-7% of the original images. The performance of VGG-Face is not as good as the other datasets, mainly due to its large scale and the corresponding complexity of the model. It is easier to generate APs on more complicated models, so it is more difficult to reconstruct the real triggers.

Position of Triggers. To evaluate the impact of the position of a trigger, we inject triggers at different positions of the input images, e.g., different corners, and randomly selected edges, into all the datasets. In total, we get 35 backdoored models and evaluate NeuralSanitizer on them. We achieved 2.0% FNR and 1.0% FPR on average. The results show that the backdoor detection of NeuralSanitizer is not affected by the positions of the triggers, since it does not make any assumption about the specific position of triggers.

Pattern and Shape of Triggers. We also evaluate the impact of different patterns and shapes of a trigger on NeuralSanitizer, e.g., blocks with different colors, more complicated patterns like a figure of Hello Kitty, square, circle, triangle, pentagram, etc. Finally, we get 32 backdoored models, with 16 models for the pattern evaluation and 16 models for the shape evaluation. We achieved 2.2% FNR and 0.8% FPR on average. Experimental results also show that NeuralSanitizer is not affected by any of the above two properties of the triggers.

Connectivity of Triggers. The pattern of a trigger can be one single piece or several pieces together. We evaluate whether a trigger with different numbers of pieces will impact the performance of our approach. We perform the following experiments: (1) a single square trigger, (2) a trigger of two squares, and (3) a trigger of four squares. The results show that NeuralSanitizer is not impacted even if the trigger contains four pieces. Interestingly, sometimes NeuralSanitizer reconstructs several pieces of the trigger, but not all. We then use the reconstructed "partial" trigger (e.g., two pieces of four pieces) to test the backdoored model and find that it is sufficient to let the model misclassify. Therefore, we believe the partial trigger reconstructed by NeuralSanitizer contains the critical features of the trigger learned by the model. In addition, the reconstructed "partial" trigger has little influence on the results of backdoor removal, i.e., NeuralSanitizer can still remove this kind of trigger effectively.

Concurrency of Backdoors. Attackers can inject more than one backdoor into a model, so we evaluate whether the number of backdoors impacts the detection of NeuralSanitizer. We consider the following two situations: (1) multiple backdoors with different target labels, which means each trigger, when activated, causes the backdoored model to misclassify the IATs into a different label; (2) multiple backdoors with the same target label, which means that all different triggers will let the model misclassify the IATs into the same target label.

In the first situation, we inject one backdoor targeting each label using the datasets MNIST, GTSRB, and CIFAR-10. Totally, there are 10, 43, and 10 backdoors targeting 10, 43, and 10 different labels for each of the datasets respectively. Since there are too many labels (1,595 and 2,622) in YouTube-Face and VGG-Face datasets, it is impractical to design as many triggers as the number of labels (each trigger should be different than the other. Otherwise, the model cannot distinguish and misclassify them into different labels). Therefore, for each of the two datasets, we train three models and inject 10, 20, and 40 different backdoors for each model. After generating the backdoored models, we evaluate NeuralSanitizer on them and achieve 2.4% FNR and 0.7% FPR on average. The results show that NeuralSanitizer can still detect all the backdoors for different target labels, because NeuralSanitizer detects the existence of triggers by traversing each label of the model, thus it is not affected by the number of infected labels.

In the second situation, for each of the datasets, we first inject 1-5 backdoors targeting one randomly chosen label and obtaining 20 such backdoored models. Then we perform NeuralSanitizer on each label recursively. In particular, for each label, if NeuralSanitizer detects a backdoor, it first removes the backdoor and then performs the second round of reconstruction. The search for each label stops when no trigger is reconstructed. Interestingly, we find that after backdoor removal using the identified trigger, some of the other unidentified triggers may no longer work, since the fine-tuning operation may also weaken the other backdoors accordingly (but will not reduce the classification accuracy on clean data). This actually is in favor of us since the model can become clean even faster. Overall, for all the models, we find none of the injected backdoors can work anymore after removal. It is noteworthy that even if attackers generate backdoors by combining the above two situations (injecting more than one backdoor for each of the target labels), NeuralSanitizer can still detect and remove all the backdoors completely by traversing each label recursively after one backdoor is reconstructed.

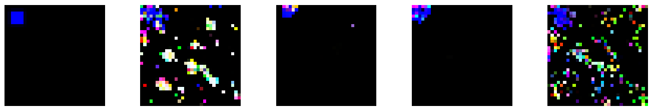
Overall, our evaluation demonstrates that NeuralSanitizer is not affected by these properties of the triggers, like the position, pattern, shape, connectivity, and concurrency.

C. Contribution of the Proposed Techniques

We evaluate the contribution of the proposed techniques, including potential trigger reconstruction, critical features preservation, and TNDA, on the final backdoor detection performance. In particular, we choose one of the proposed techniques each time, either remove it or replace it with the technique used in related works, and measure the backdoor detection performance.

Potential Trigger Reconstruction. To evaluate the contribution of our potential trigger reconstruction, we replace it with the trigger reconstruction approach proposed by Neural Cleanse, and still utilize our TNDA to identify the real triggers. The experimental results show that the FPR is similar to that of NeuralSanitizer, but FNR increases from 2.1% to 17.4%, which can be explained as below. The approach proposed in NC needs to minimize the size of triggers when reconstructing them, thus hardly reconstructing large triggers. Therefore, no trigger will be reconstructed by the approach in NC when larger triggers were embedded into the model, which eventually leads to false negatives since there is no real trigger for TNDA to identify. In contrast, our potential trigger reconstruction can also reconstruct large triggers, resulting in smaller FNR.

Critical Features Preservation. As in Section V-A, after reconstructing the potential triggers, a mix of the real trigger and some other irrelevant features could be generated together with the real trigger and universal APs. If we do not remove the irrelevant features from the mix, it will pass through TNDA and cannot be distinguished from universal APs correctly. To verify this assumption, we measure FPR and FNR when the irrelevant features are not removed. On average, FPR is similar, but FNR increases to 11.7%, compared with 2.1% when removing those irrelevant features. The results confirm our assumption that some features of APs in the mix help it to transfer to



(a) Trigger (b) $0.1 \cdot \alpha$ (c) $10 \cdot \alpha$ (d) $10 \cdot \beta$ (e) $0.1 \cdot \beta$

Fig. 1: Original and reconstructed triggers

the tuned models even though the trigger itself inside the mix cannot transfer. Therefore, critical features preservation facilitates TNDA to distinguish real triggers from universal APs.

TNDA. NeuralSanitizer proposes Transferability-based Neural Differential Analysis (TNDA) to identify real triggers from APs. To evaluate its contribution, we use our approach to reconstruct potential triggers, and replace TNDA with the approach in Neural Cleanse to identify real triggers. The evaluation results show that the FPR is similar to NeuralSanitizer, but FNR increases from 2.1% to 18.1%. This is reasonable because the approach proposed in NC relies on the size of the trigger to identify the real trigger from APs, i.e., NC identifies a reconstructed pattern as the trigger if the pattern is much smaller than other reconstructed patterns. When the original trigger is large, the reconstructed trigger is also relatively large, similar to or even larger than APs. Therefore, NC will fail to identify the real trigger from APs but identify all of them as APs, thus resulting in a high FNR.

D. *Influence of Hyperparameters*

Reconstructing potential triggers. There are two important weights in reconstructing potential triggers, α, and β in the objective function. When doing the experiment, we change the value of one weight while keeping another unchanged. Specifically, we increase the value by 10%, 20%, 50%, and 100%, or decrease the value by 10%, 20%, 30%, and 50%, and find that the reconstructed patterns are just slightly changed. We then increase the scope of the changes, i.e., 10 times, and find that the reconstructed pattern will change obviously as shown in Figure 1, where Figure 1 (a) is the original trigger and others are the reconstructed patterns. When the value of α decreases by 90%, the attack success rate of the reconstructed pattern is still very high (almost 100%), but the reconstructed pattern becomes very large with scattered features as shown in Figure 1 (b). When the value of α increases by 10 times, the attack success rate becomes relatively low (about 80%), and a very small pattern is reconstructed as shown in Figure 1 (c). Changing the value of β will influence the connectivity of the pattern. A very large value makes the pattern more connected as shown in Figure 1 (d), while a very small value generates a pattern with noise everywhere as shown in Figure 1 (e). We also evaluate the influence of n and v in the objective function. The number of images n has little influence as long as n is not too small (i.e., smaller than 10). Changing the value of v will influence the distribution of the classification results. We find that setting its value to n divided by the number of labels can make the classification results roughly follow a uniform distribution.

TABLE VIII: Influence of the threshold to identify random patterns

Threshold	0.4	0.5	0.6	0.7	0.8
FNR	1.9%	1.9%	2.1%	2.9%	3.5%
FPR	9.7%	4.0%	0.9%	0.2%	0.0%

TABLE IX: Influence of l in TNDA

l	1	2	3	4	5	6
FNR	1.5%	1.5%	1.8%	2.0%	2.0%	2.1%
FPR	3.9%	3.0%	2.0%	1.6%	1.2%	0.9%

TABLE X: Influence of q in TNDA

q	1	2	3	4	5	6
FNR	2.1%	1.4%	0.7%	0.5%	0%	0%
FPR	0.9%	1.8%	2.8%	4.0%	4.6%	5.1%

TABLE XI: Influence of the threshold in TNDA for MNIST and CIFAR-10

Threshold	0.2	0.3	0.4	0.5	0.6
FNR	8.2%	3.7%	1.8%	0.8%	0%
FPR	0.8%	1.1%	1.5%	3.4%	4.7%

TABLE XII: Influence of the threshold in TNDA for GTSRB, Youtube-Face, VGG-Face, TS and TW

Threshold	0.1	0.15	0.2	0.25	0.3
FNR	5.2%	4.9%	2.4%	1.9%	1.5%
FPR	0.05%	0.07%	0.08%	1.0%	1.1%

Preserving critical features. An important hyperparameter introduced here is the attack success rate threshold used to identify random patterns (Case IV in Section V-A). We change its value from 40% to 80% and the results are shown in Table VIII. We find that the selection of the threshold value is a trade-off between FNR and FPR. A smaller threshold decreases FNR but increases FPR, while a larger threshold decreases FPR but increases FNR. Therefore, we use 0.6 as the threshold value to balance FNR and FPR in all other experiments. Another hyperparameter used here is the threshold to control when preserving critical features should stop, i.e., when the attack success rate of the remaining trigger drops below the threshold. We change its value from 0.8 to 1 and find that the optimal range is from 0.9 to 0.98. A smaller value removes too many features (causing a low attack success rate), while a larger value keeps too many irrelevant features.

TNDA. There are two important hyperparameters in TNDA, i.e., l and q. We change the value of one and keep another unchanged in the following experiments. The results are shown in Tables IX and X. Generally, l has little influence on FNR. However, its increase causes FPR to decrease, since IATs can hardly transfer to any of the tuned models but more tuned models can increase the probability of AEs' transferability to one of them, thus reducing FPR. The selection of q is a trade-off between FNR and FPR. A smaller value identifies more APs, but some triggers may also be misidentified occasionally, thus causing relatively low FPR but high FNR. In contrast, a larger value brings low FNR but high FPR.

Another hyperparameter in TNDA is the threshold of transferability. We consider a potential trigger can transfer to a model when its attack success rate on this model is higher than such a threshold. The setting of the threshold is related to the number of labels of the models. For MNIST, GTSRB, and CIFAR-10 with fewer labels, we use a larger threshold, i.e., 0.4, since the proportion of even a set of clean images

TABLE XIII: Influence of k in PNNIR

k	1	2	3	4	5	6
ASR	4.3%	4.0%	3.8%	3.2%	3.4%	3.0%
ACC	93.7%	93.4%	89.6%	85.1%	81.3%	78.2%

ACC: Accuracy on clean data, ASR: attack success rate of backdoor.

TABLE XIV: Influence of learning rate

Learning rate		10^{-2}	10^{-3}	10^{-4}	10^{-5}
PNNIR	ACC	35.7%	58.2%	86.3%	76.4%
	ASR	0.9%	1.2%	1.7%	3.1%
Backdoor removal	ACC	84.5%	88.2%	93.8%	94.3%
	ASR	1.7%	2.4%	3.0%	3.4%

ACC: Accuracy on clean data, ASR: attack success rate of backdoor.

TABLE XV: Influence of ratio of clean data

Ratio of clean data		1%	2%	5%	10%
PNNIR	ACC	77.2%	82.3%	83.7%	86.5%
	ASR	2.2%	1.7%	1.4%	1.6%
Backdoor removal	ACC	93.1%	93.4%	94.2%	94.3%
	ASR	3.0%	2.4%	2.8%	3.1%
FNR		2.4%	2.2%	2.0%	2.1%
FPR		1.5%	1.2%	1.0%	0.9%

ACC: Accuracy on clean data, ASR: attack success rate of backdoor.

classified into each class is relatively high given fewer labels exist. In contrast, for other models with thousands of labels, we use a smaller threshold, i.e., 0.2. We change the values of the threshold for the two types of models and the results of FNR and FPR are shown in Table XI and Table XII respectively. We find that the selection of the threshold value is also a trade-off between FNR and FPR. A smaller threshold will identify more APs, but some triggers may also be misidentified occasionally, thus causing low FPR and high FNR. On the contrary, a larger threshold will cause low FNR and high FPR.

PNNIR and backdoor removal. There is one hyperparameter used in PNNIR, i.e., the number of initialized layers *k*. We adopt three representative backdoor attack methods, i.e., Badnets [1], TrojanNet [2], and Latent Backdoor [4], on all five datasets, i.e., MNIST, CIFAR-10, GTSRB, Youtube-Face, and VGG-Face, and evaluate the influences of *k* on the backdoor attack success rate (ASR) as well as the clean data classification accuracy (ACC). We change the value of *k* from one (the least) to six (all layers for MNIST) on all datasets and show the average results in Table XIII. The ACC decreases as the increase of *k*, which is reasonable since more layers are initialized. We also find that initializing only one layer (*k = 1*) is generally enough to cause backdoors to fail.

We also evaluate the influence of the learning rate and the ratio of the clean data to the whole training dataset on the backdoor attack success rate (ASR) as well as the clean data classification accuracy (ACC). The results are shown in Table XIV and Table XV respectively. We find that both the learning rate and the ratio of clean data influence the ACC, but have little influence on the ASR. A too large or a too small learning rate reduces the ACC, so we simply choose a learning rate with the highest ACC. A smaller ratio indicates a fewer amount of the training data, which causes the model to over-fit easily to achieve lower ACC. The ACC of PNNIR is lower than that of backdoor removal, which is reasonable because PNNIR initializes some layers but backdoor removal does not. The ACC of PNNIR drops from 86.5% to 77.2% when the ratio decreases from 0.1 to 0.01, while backdoor removal still maintains a high ACC even if only 1% of the training dataset is used. We also evaluate the influence of the ratio of clean data on the end-to-end backdoor detection performance, i.e., the FNR and FPR. As shown in Table XV, NeuralSanitizer can still maintain low FNR and FPR even when only 1% of clean data is available, which indicates the drop of ACC for tuned models has little influence on the transferability.

E. Extension to Speech Recognition Models

Since the two fundamental properties of triggers and Lemma 1 are generic, not just limited to the vision domain, it is feasible to extend NeuralSanitizer to other domains by slightly modifying the trigger reconstruction scheme. We use the speech recognition domain as an example and extend NeuralSanitizer to verify its effectiveness in this domain. Different from the vision domain where the DNN models are usually trained on raw images, speech recognition models are usually trained on the features extracted from the raw audios. For instance, Mel-Frequency Cepstral Coefficients (MFCC) is one of the most commonly used speech feature extraction algorithms. Therefore, we design an objective function to reconstruct potential triggers on the spectrogram generated by MFCC. The spectrogram is a two-dimensional matrix, with each row representing a time frame of the audio and each column containing the features extracted from the corresponding time frame.

The objective function used to reconstruct the potential triggers is similar to that used in Section IV-C with two major differences: (1) both Δ containing the features of the reconstructed trigger and m controlling the location of the reconstructed trigger are 2D matrices, instead of 3D matrices; (2) the noise reduction function L_{noise} is applied to both Δ and m, rather than to m only:

$$L_{noise} = \sum_{j,k} \sum_{a=-1}^{1} (|m_{j,k} - m_{j+a,k}| + |\Delta_{j,k} - \Delta_{j,k+a}|) \quad (3)$$

Regarding the first difference, since the spectrogram generated by MFCC is a 2D matrix, Δ and m need to have the same dimension as the spectrogram. The second difference is related to the property of MFCC. After multiple operations of MFCC, e.g., Fourier transform and discrete cosine transform, the adjacent values in the spectrogram are usually close to each other. Therefore, to reconstruct the trigger Δ in the spectrogram, its adjacent values are also constrained by the noise reduction function L_{noise}. The following steps to detect backdoors in the speech recognition domain are similar to those to detect patch-based triggers.

We evaluate NeuralSanitizer on the open-source speech recognition backdoored models provided in TrojanNN [2], which are used to recognize spoken numbers in English. Any speech synthesized with the trigger will be recognized as the target number by those backdoored models. Based on our evaluation, NeuralSanitizer achieves 2.5% FNR and 2.7% FPR on average, similar to the performance in the vision domain.

F. Implementation Details

In this section, we detail the implementation of NeuralSanitizer.

Input Preprocessing. For the MNIST, GTSRB, and Youtube-Face datasets, the pixel values of raw images (belonging to

[0, 255] are first subtracted by 127.5, then divided by 255, thus normalized to [-0.5, 0.5]. For the CIFAR-10 dataset, the pixel values of raw images (belonging to [0, 255]) are first subtracted by the mean value (i.e., 120.7) and then divided by the standard deviation (i.e., 64.15). For the VGG-Face dataset, we follow the same preprocessing method used in [2], i.e., for the raw images (belonging to [0, 255]) with the RGB channels, the pixel values in R channel are subtracted by 129.1863, the pixel values in G channel are subtracted by 104.7624, and the pixel values in B channel are subtracted by 93.5940.

PNNIR. In Section IV-B, we propose Partial Neural Network Initialization and Retraining (PNNIR) to generate tuned models by first initializing the last one to three layers and retraining the model. For models with parallel layers such as DeepID and Resnet18, we treat the block composed of these parallel layers as one layer for initialization. We initialize the selected layers with random numbers, which follow a uniform distribution from $[-10^{-1}, 10^{-1}]$. When retraining the DNN model, we first freeze all other layers and only train the initialized layers, and then train the whole DNN model. We adopt Adam [3] optimizer to train the DNN model. The learning rate is set as 10^{-3} on small datasets (i.e., MNIST, GTSRB and CIFAR10) and 10^{-4} on large datasets (i.e., Youtube-Face). In each iteration of training, we randomly select 100 images from the clean dataset as a mini-batch to train the model. The model training ends when the model converges, i.e., the performance of the model becomes stable.

Generating Potential Triggers. We optimize the objective function proposed in Section IV-C based on gradients. Adam optimizer [3], an iterative optimizer, is used to slightly modify Δ and m in each iteration. The number of iterations is set as 100, which is enough for reconstructing real triggers in most cases. The learning rate is set as 0.1, 0.1, 0.1, 0.1, and 2.5 on MNIST, GTSRB, Youtube-Face, CIFAR-10, and VGG-Face respectively. The reason why the learning rate on the VGG-Face dataset is significantly larger than other datasets is that the input range on the VGG-Face dataset is much larger ([0,255] compared to that of other datasets [0,1]). We set $n = 10$, $\alpha = 1$, and adjust β to make the pattern of m not too noisy. The values of β are 0.005, 0.005, 0.001, 0.005, and 0.0001 on MNIST, GTSRB, Youtube-Face, CIFAR-10, and VGG-Face, respectively. We also re-sample x_i in the small clean dataset evenly in each iteration. A relatively larger value is used for the vector v on the datasets with fewer labels, i.e., 2 on MNIST and CIFAR-10, and a smaller value, i.e., 1 on other datasets with more labels.

Critical Features Preservation. For each potential trigger, we randomly select 100 images from the clean dataset and attach the trigger to each of them. We then adopt Grad-CAM to get the heat map for each image attached by the trigger and compute the average heat map. We gradually remove less "significant" pixels from the potential trigger according to their corresponding values in the average heat map. Specifically, we define t, a threshold used to remove the less "significant" pixels. We remove all the pixels from the reconstructed trigger pattern if their values on the average heat map are less than the threshold t. The initial value of t is set as 0 and increased by 0.01 in each iteration. The above iteration stops until the attack success rate of the manipulated potential trigger (with those less "significant" pixels removed) drops below a pre-defined threshold, i.e., 95% of the original attack success rate.

TNDA. For each potential trigger, we randomly select 100 images from the clean dataset, attach the trigger to each of them, and test whether the attack success rate is higher than a certain threshold. The setting of the threshold is related to the number of labels of the datasets. For MNIST and CIFAR-10 with few labels, we use a higher threshold value, e.g., 40%, since when the number of labels is small, the proportion of a set of clean images classified into a wrong class cloud be relatively high. In contrast, for other datasets with more labels, we use a lower threshold, e.g., 20%. If the potential trigger can transfer to at least $q = 1$ tuned model, we classify it as an AP, otherwise, as the real trigger.

Removing Backdoors. For each reconstructed trigger, we randomly select 10% of the images from the clean dataset, place the reconstructed trigger onto these images, and label them correctly. Then we put the correctly labeled synthetic images back into the clean dataset to fine-tune the whole backdoored model, thus eliminating the correlation between the triggers and their target label to remove the backdoor. We adopt Adam [3] optimizer to train the backdoored model. The learning rate is set as 10^{-4} on the MNIST, GTSRB, and Youtube-Face datasets, and 10^{-5} on the CIFAR-10 and VGG-Face datasets. In each iteration of training, we randomly select 100 images from the clean dataset as a mini-batch to train the model. The number of iterations is set as 2000, which is enough to make the model converge.

Dataset Statistics and Model Structure. Details of the five datasets and model structures are shown in Tables XVI, XVII, XVIII, XXI, XX, and XIX.

TABLE XVI: Dataset Statistics

Dataset	Labels	Image Size	Training Images
MNIST	10	28x28x1	50000
GTSRB	43	32x32x3	35288
CIFAR-10	10	32x32x3	50000
YouTube-Face	1595	224x224x3	621100
VGG-Face	2622	224x224x3	2622000

TABLE XVII: Model Architecture of MNIST

Layer type	Filter size	Channels	Stride	Activations
Conv	3×3	32	1	ReLU
Conv	3×3	32	1	ReLU
MaxPool	2×2	32	2	-
Conv	3×3	64	1	ReLU
Conv	3×3	64	1	ReLU
MaxPool	2×2	64	2	-
FC	-	512	-	ReLU
FC	-	10	-	Softmax

TABLE XVIII: Model Architecture of GTSRB

Layer type	Filter size	Channels	Stride	Activations
Conv	3×3	32	1	ReLU
Conv	3×3	32	1	ReLU
MaxPool	2×2	32	2	-
Conv	3×3	64	1	ReLU
Conv	3×3	64	1	ReLU
MaxPool	2×2	64	2	-
Conv	3×3	128	1	ReLU
Conv	3×3	128	1	ReLU
MaxPool	2×2	128	2	-
FC	-	512	-	ReLU
FC	-	43	-	Softmax

TABLE XIX: Model Architecture of VGG-Face

Layer type	Filter size	Channels	Stride	Activations
Conv	3×3	64	1	ReLU
Conv	3×3	64	1	ReLU
MaxPool	2×2	64	2	-
Conv	3×3	128	1	ReLU
Conv	3×3	128	1	ReLU
MaxPool	2×2	128	2	-
Conv	3×3	256	1	ReLU
Conv	3×3	256	1	ReLU
Conv	3×3	256	1	ReLU
MaxPool	2×2	256	2	-
Conv	3×3	512	1	ReLU
Conv	3×3	512	1	ReLU
Conv	3×3	512	1	ReLU
MaxPool	2×2	512	2	-
Conv	3×3	512	1	ReLU
Conv	3×3	512	1	ReLU
Conv	3×3	512	1	ReLU
MaxPool	2×2	512	2	-
FC	-	4096	-	ReLU
FC	-	4096	-	ReLU
FC	-	2622	-	Softmax

TABLE XX: Model Architecture of YouTube-Face

Layer name	Layer type	Filter size	Channels	Stride	Activations	Link to
conv1	Conv	4×4	20	2	ReLU	INPUT
pool1	MaxPool	2×2	20	2	-	conv1
conv2	Conv	3×3	40	2	ReLU	pool1
pool2	MaxPool	2×2	40	2	-	conv2
conv3	Conv	3×3	60	2	ReLU	pool2
pool3	MaxPool	2×2	60	2	-	conv3
conv4	Conv	2×2	80	1	ReLU	pool3
fc1	FC	-	160	-	-	pool3
fc2	FC	-	160	-	-	conv4
add1	ADD	-	-	-	ReLU	fc1,fc2
fc3	FC	-	1595	-	Softmax	add1

TABLE XXI: Model Architecture of CIFAR-10

Layer name	Layer type	Filter size	Channels	Stride	Activations	Link to
conv0	Conv	3×3	64	1	ReLU	INPUT
conv1-1	Conv	3×3	64	1	ReLU	conv0
conv1-2	Conv	3×3	64	1	ReLU	conv1-1
add1	ADD	-	-	-	-	conv0,conv1-2
conv2-1	Conv	3×3	64	1	ReLU	add1
conv2-2	Conv	3×3	64	1	ReLU	conv2-1
add2	ADD	-	-	-	-	conv1-2,conv2-2
conv3-1	Conv	3×3	128	2	ReLU	add2
conv3-2	Conv	3×3	128	1	ReLU	conv3-1
conv3-3	Conv	1×1	128	2	-	conv2-2
add3	ADD	-	-	-	-	conv3-3,conv3-2
conv4-1	Conv	3×3	128	1	ReLU	add3
conv4-2	Conv	3×3	128	1	ReLU	conv4-1
add4	ADD	-	-	-	-	conv3-2,conv4-2
conv5-1	Conv	3×3	256	2	ReLU	add4
conv5-2	Conv	3×3	256	1	ReLU	conv5-1
conv5-3	Conv	1×1	256	2	-	conv5-1
add5	ADD	-	-	-	-	conv5-3,conv5-2
conv6-1	Conv	3×3	256	1	ReLU	add5
conv6-2	Conv	3×3	256	1	ReLU	conv6-1
add6	ADD	-	-	-	-	conv5-2,conv6-2
conv7-1	Conv	3×3	512	2	ReLU	add6
conv7-2	Conv	3×3	512	1	ReLU	conv7-1
conv7-3	Conv	1×1	512	2	-	conv7-1
add7	ADD	-	-	-	-	conv7-3,conv7-2
conv8-1	Conv	3×3	512	1	ReLU	add7
conv8-2	Conv	3×3	512	1	ReLU	conv8-1
add8	ADD	-	-	-	-	conv7-2,conv8-2
pool	AvgPool	4×4	512	1	-	add8
fc	FC	-	10	-	Softmax	pool

REFERENCES

[1] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, "Badnets: Evaluating backdooring attacks on deep neural networks," *IEEE Access*, vol. 7, pp. 47 230–47 244, 2019.

[2] Y. Liu, S. Ma, Y. Aafer, W. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," in *25th Annual Network and Distributed System Security Symposium*, 2018.

[3] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations*, 2015.

[4] Y. Yao, H. Li, H. Zheng, and B. Y. Zhao, "Latent backdoor attacks on deep neural networks," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2041–2055.