Better Trigger Inversion Optimization in Backdoor Scanning

Guanhong Tao, Guangyu Shen, Yingqi Liu, Shengwei An, Qiuling Xu
Shiqing Ma†, Pan Li, Xiangyu Zhang
Purdue University, †Rutgers University
{taog, shen447, liu1751, an93, xu1230, panli, xyzhang}@cs.purdue.edu
†sm2283@cs.rutgers.edu

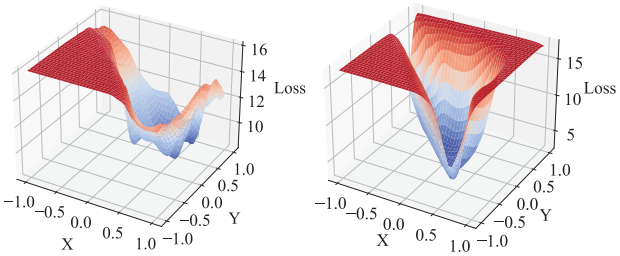
Abstract

Backdoor attacks aim to cause misclassification of a subject model by stamping a trigger to inputs. Backdoors could be injected through malicious training and naturally exist. Deriving backdoor trigger for a subject model is critical to both attack and defense. A popular trigger inversion method is by optimization. Existing methods are based on finding a smallest trigger that can uniformly flip a set of input samples by minimizing a mask. The mask defines the set of pixels that ought to be perturbed. We develop a new optimization method that directly minimizes individual pixel changes, without using a mask. Our experiments show that compared to existing methods, the new one can generate triggers that require a smaller number of input pixels to be perturbed, have a higher attack success rate, and are more robust. They are hence more desirable when used in real-world attacks and more effective when used in defense. Our method is also more cost-effective.

1. Introduction

Backdoor attacks aim to induce model misclassification of arbitrary input samples to a target label by stamping a special input pattern called *trigger*. Backdoors could be *injected* by various methods, such as data poisoning [17, 35, 40] and neuron hijacking [39], and also naturally exist in normally trained models, called *natural backdoors* [41]. The latter is caused by distribution bias of low level features and can be exploited just like injected backdoors. For example, if a person always wears a unique pair of glasses in a clean face recognition dataset, the glasses may become a trigger to induce misclassification to the person.

Due to the prominent threat of backdoors, researchers have proposed a large body of defense solutions (see Section 2). Among them, backdoor scanning [21, 22, 68, 76] is an important type of defense. Many scanners [38, 41, 59, 65, 75] rely on *trigger inversion*, which leverages optimization to derive a small input pattern that can flip clean samples (of a *victim class*) to the target label. A model is considered having backdoor if an exceptionally small trigger can be found.



(a) NC (b) Ours

Figure 1. Loss landscapes of NC and our method. The x-axis and y-axis show the coefficients on two random directions. The z-axis denotes the loss value.

Most existing trigger inversion methods (e.g., ABS [38], K-arm [59], and Tabor [19]) are built on Neural Cleanse (NC) [65], which decouples a trigger into a perturbation vector and a *mask*. The perturbation vector denotes the perturbations applied to an input and the mask determines which part of the perturbation vector should be applied. NC minimizes the mask and the perturbation vector together to produce a small trigger (details in Section 3.1). Due to the multiplication correlation between the mask and the perturbation vector during optimization, NC can fall into local optima and fail to reach the optimal trigger, i.e., the smallest trigger with a high attack success rate. Figure 1a shows the loss landscape of NC using the contour plot with two random directions [16, 24, 31] with $(x = 0, y = 0)$ the optimum. Observe that there are multiple dips (local optima) on the loss surface, which prevent NC from reaching the optimum. In addition, triggers by NC are often not robust and may become ineffective when undertaking transformations (see Section 5.3).

Figure 2 shows the results of various techniques for generating a natural backdoor pattern for a normally trained

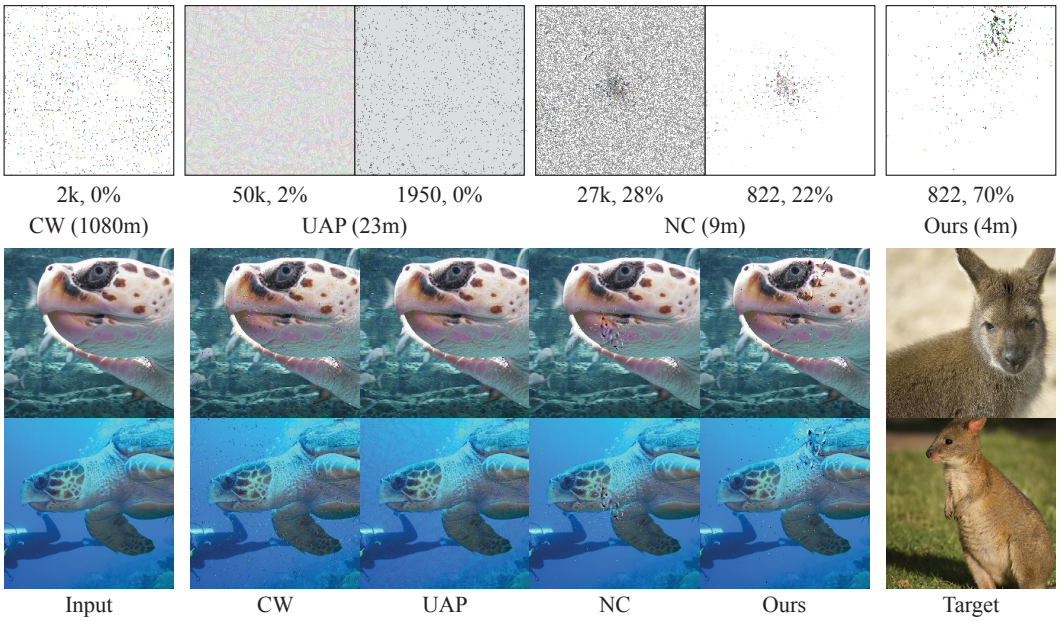


Figure 2. Comparison of generated backdoors. In the first row, the texts below backdoor images denote the number of perturbed pixels and the ASR on all the samples of loggerhead turtle from the validation set. The value shown together with the method name denotes the trigger generation time cost in minutes. The bottom two rows show example images stamped with backdoors by different methods, where the first column gives the victim class images and the last column the target class images.

model on ImageNet downloaded from [25]. Stamping each of these backdoors on sea turtle images can flip them to the kangaroo class. The first row shows the backdoor patterns by various inversion techniques. From left to right, the second and third rows show samples from the victim class (column 1), samples stamped with the backdoor patterns (columns 2-5), and the target class samples (last column). The fourth and fifth images in the first row denote the trigger generated by NC and its reduced version, respectively. Observe that the NC trigger requires perturbing 27k pixels and has only 28% ASR. When we reduce the NC trigger by removing the smallest perturbations to size 822 (which is the same as our trigger), the ASR degrades to 22%. This is because of the large number of local minima, as those on the loss surface of NC in Figure 1a. Our results in Section 5 show that on average, when NC triggers are reduced to the same size of ours, their ASRs on average degrade by 26%.

Problem Statement. In the context of backdoor attack and defense, a good optimization method (for trigger generation) is critical. In this paper, we say a method is good if it produces triggers that are (1) small (i.e., having a small number of perturbed pixels), (2) having a high *attack success rate* (ASR) (the percentage of unseen clean samples that can be flipped by the trigger), (3) robust (against input transformations), and (4) has low computation overhead. A good trigger generation method serves both attack and defense. If it is used in attack, e.g., generating natural triggers for normally trained models to induce intended misclassification, a small and robust trigger makes the attack easy to launch and effective in the physical world. If it is used in defense, smaller triggers can help scanners more effectively determine if a model is trojaned, as an exceptionally small trigger is a good indicator of injected backdoor [38, 59, 65], and have better effectiveness in model hardening. □

We propose a novel optimization method. Instead of optimizing the product of the perturbation vector and the mask, our method only optimizes a perturbation vector. Specifically, we leverage the long-tail effects of $\tanh$ function to represent the binary nature of perturbations, with one end modeling the maximum perturbation and the other end no perturbation. We introduce two $\tanh$ functions for each pixel, one denoting positive perturbation and the other negative. Our optimization method has a much smoother loss surface than NC as shown in Figure 1b. Observe that the loss values all descend along the valley towards the optimal point at the bottom. On the ImageNet dataset, our generated triggers are two orders of magnitude smaller than those of NC, 2.73 times more robust, and have 20% higher ASR on average. Our method is 2.15 times faster than NC. The last image in the first row of Figure 2 shows our trigger. It has the smallest number of perturbed pixels (822) with the highest ASR (70%) on the unseen validation set. We also compare with UAP [58] and CW [4] (another two trigger generation methods adapted from adversarial attacks). Ours is one or more orders of magnitude faster. The implementation of our method is publicly available [1].

2. Related Work

Backdoor Attack. Existing backdoor attacks poison the training set using intentionally crafted samples with injected backdoor patterns together with the target label such as patch attacks [7, 17]. To achieve stealthiness, a different type of backdoor attacks applies imperceptible perturbations on poisoned data with the original label like clean label attacks [55, 57, 78]. Another type of backdoor attacks crafts different backdoors for different inputs [35,49,56]. Other than poisoning the training set, backdoors also naturally exist in clean models [41]. Backdoor attacks can be launched on models with various applications, such as natural language processing [28, 77], transfer learning [53, 66, 73], and federated learning [3, 67, 71].

Backdoor Defense. To detect poisoned models [19, 22, 26, 52, 72], existing works reverse-engineer backdoors [38, 65], and leverage the difference between poisoned and clean models when reacting to input perturbations [21, 68, 76]. Existing techniques also detect and reject inputs stamped with backdoors [5, 6, 8, 10, 12, 13, 34, 42, 43, 60, 62, 63]. Verification methods aim to provide guarantees that models are not vulnerable to certain types of backdoors [23, 30, 64, 69]. There are also works focusing on eliminating backdoors [33] by pruning out compromised neurons [37] or retraining leveraging data augmentation technique [74].

Optimization Methods for Trigger Generation. NC [65] is a state-of-the-art and we have detailed discussion and comparison throughout the paper. Existing adversarial attack methods were proposed to generate per-instance perturbations, such as fast gradient sign method (FGSM) [15], projected gradient descent (PGD) [44], JSMA [51], CW [4], and SLIDE [61], etc. Universal adversarial perturbation (UAP) [46] aims to generate a global perturbation that can cause a set of inputs to misclassify. We extend some of them to generate triggers (see the following section).

3. Existing Optimization Methods for Backdoor Trigger Generation and Their Limitations

In this section, we discuss in detail NC's optimization in trigger generation and two other optimizations that are popular in adversarial attack and hence adapted to trigger generation (i.e., CW and UAP). We focus on studying their limitations in trigger generation.

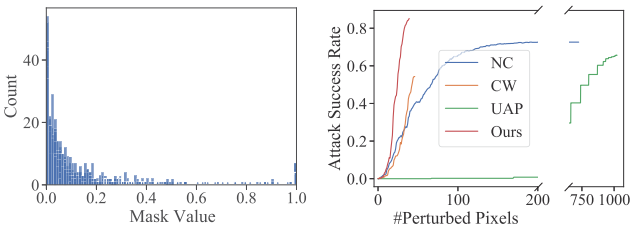
3.1. Optimization of Neural Cleanse (NC)

As mentioned earlier, the optimization method in NC is the most popular in trigger generation. Specifically, it solves the following optimization problem.

$$\min_{\boldsymbol{m},\boldsymbol{p}} \mathcal{L}_{\text{NC}} = \mathcal{L}\big(\mathcal{M}(\boldsymbol{x}'), y_t\big) + \lambda \cdot \|\boldsymbol{m}\|_1, \forall \boldsymbol{x} \in \boldsymbol{X}, \quad (1)$$

$$\text{where } \boldsymbol{x}' = (1-\boldsymbol{m}) \circ \boldsymbol{x} + \boldsymbol{m} \circ \boldsymbol{p}. \quad (2)$$

Variables $\boldsymbol{m}$ and $\boldsymbol{p}$ denote the mask and the perturbation vector, respectively; $\mathcal{L}(\cdot,\cdot)$ denotes the cross entropy loss function of the subject model $\mathcal{M}$; y_t is the target label. Intuitively, the optimization aims to flip the classification result (the first term in the loss function) and reduce the trigger size (the second term). The introduction of the mask enables using optimization to reduce the trigger size. However, it also has some undesirable effects. NC has to optimize both $\boldsymbol{m}$ and $\boldsymbol{p}$ that are correlated by the $\circ$ operation in Equation 2, which is difficult and leads to low ASRs and large sizes. NC tends to produce many small values in the mask, indicating the corresponding input pixels need to be slightly perturbed. Although these values are small, many of them cannot be set to zero. Otherwise, the ASR degrades. These small and pervasive perturbations make attack in the physical world difficult and the backdoor not robust against input transformations (see results in Section 5.3).



(a) The distribution of mask values for 10 runs of NC

(b) The relation between ASR and perturbed pixels

Figure 3. Characteristics of generated backdoors on all the test samples of a victim class from the CIFAR-10 dataset

Figure 3a shows the distribution of mask values for 10 random runs of NC for generating a natural trigger that flips plane to dog in a ResNet20 model on CIFAR-10. Observe that a large portion of mask values fall in the range from 0 to 0.1, which is equivalent to keeping 90% of the original pixel values. In Figure 3b, we start from the generated triggers, gradually set the smallest mask values to 0, which is equivalent to gradually reducing the number of perturbed pixels, and show the changes of ASR with the number of perturbed pixels. The number of perturbed pixels of the NC trigger is 725 with the ASR of 0.66. According to Figure 3a, most of them have small values. However, when the number of perturbed pixels is gradually reduced to 150, the ASR starts to degrade quickly, indicating the perturbations on these pixels need to be retained, even though they are still small. In contrast, our trigger has 0.83 ASR with only 39 perturbed pixels. Besides making physical attack difficult and not robust, the larger triggers by NC are less effective in expos-

ing injected pervasive backdoors and model hardening (see Section 5.4 and Section 5.5).

3.2. Optimization of CW

There are existing optimization methods in adversarial attack that can be adapted for trigger generation, such as JSMA [51] and CW [4], with the later the state-of-the-art. The CW L^0 attack first searches for perturbations on all pixels that can cause misclassification using the L^2 norm. It then uses a processing step external to the optimization to remove the perturbations that are the least important after each optimization epoch. The algorithm can be easily adapted to generate backdoor: instead of optimizing one input, we optimize a set of inputs. Details of the optimization can be found in Appendix A.

The adapted CW optimization has a few limitations in trigger generation. First, it is very expensive. To determine the unimportant perturbations, it has to perform gradient back-propagation to each pixel of each input and sort the importance values for all pixels. As a result, it is often two to three orders of magnitude slower than our technique and NC (see Section 5). Second, its trigger size reduction is by an external step instead of optimization, and the reduction is monotonic. As such, if some step of reduction is not towards a global minimal, it cannot be reverted. As a result, CW's optimization yields 28.28% lower ASRs and 17.25% larger trigger sizes on average compared to ours on CIFAR-10 (see Section 5). The first image in the first row of Figure 2 shows a CW backdoor. Its number of perturbed pixels (2k) is smaller than NC (27k) but larger than ours (822). However, its ASR is close to 0. It also takes 1,080 minutes to generate, compared to 4 minutes in our method.

3.3. Optimization of Universal Adversarial Perturbation (UAP)

UAP [46, 58] generates a global perturbation that can cause a set of inputs to misclassify. It has a similar goal as ours and can be adapted for trigger generation. Details can be found in Appendix B.

4. Our Method

According to our problem statement in the introduction section, having a small number of perturbed pixels is critical for backdoor trigger generation. NC uses a mask vector to denote which parts of an input are subject to perturbation. However, it requires optimizing the product of the mask and the perturbation vector, which is difficult. We propose to directly optimize a perturbation vector, without using a mask like that in NC and CW. We use $\tanh$ functions to denote perturbations of individual pixels and use optimization to minimize the sum of all these functions. The long-tail effects of the $\tanh$ function allow us to nicely model the two ends for a pixel's value change, namely, a pixel is either not

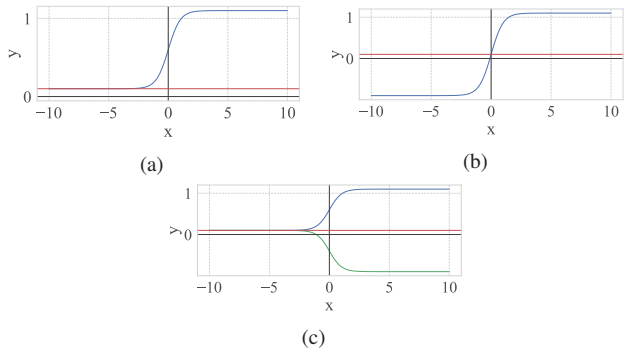


Figure 4. Illustration of using different $\tanh$ functions for the perturbation of a pixel. In (a), we denote positive change by adding $\frac{1}{2}(\tanh(x)+1)$ to the original pixel value (red line). In (b), we denote both positive and negative changes by adding $\tanh(x)$ to the pixel value. In (c), we use two $\tanh$ functions to denote positive and negative changes, respectively.

changed at all or has change of arbitrarily large magnitude (within bound). Figure 4a illustrates the concept. The y axis denotes pixel value and the x axis perturbation. The former is normalized to $[-1, 1]$ and the latter is in $(-\infty, +\infty)$. The red horizontal line denotes an original pixel value. The blue curve denotes how the pixel value changes with x. The pixel is changed by adding $\frac{1}{2}(\tanh(x)+1)$. Note that although x is unbounded, the $\tanh$ function bounds the pixel value change in $(0, 1)$. Observe that the long left tail of the blue curve means that a large number of x values on the left correspond to close-to-0 changes to the pixel, whereas the long right tail means that those x values on the right correspond to the maximum change. The shape and the continuity of the curve on one hand encourage achieving tail values (in order to have a small loss value), and on the other hand, allow perturbations to recover from tail values if needed.

However, using one $\tanh$ for each pixel only allows denoting changes along one direction, positive or negative. A naïve design is to use one tail to denote maximum positive change and the other tail to denote maximum negative change. That is, the pixel is changed by $\tanh(x)$. However, it loses the key benefit of encouraging as many pixels to have 0 value change as possible. Figure 4b illustrates the concept. Observe that the blue curve tends to go to either the maximum positive or the maximum negative. The part denotes 0 change (i.e., the interaction of the blue curve and the red line) has a steep slope such that it is unlikely for the optimization to stabilize at this point. Our solution is hence to use two $\tanh$ functions for a pixel, one denoting positive change and the other negative. Figure 4c illustrates the concept. In addition to the blue curve going upward, there is also the green curve that goes downward, denoting the negative changes. The key difference from the above naïve method is that both curves have a long tail on zero change,

which enables the optimization to stabilize. If the optimization desires positive change, it just needs to go up along the blue curve and stay on the left tail along the green curve, and vice versa. Formally, we have the following optimization objectives.

$$\min_{\boldsymbol{b}_p, \boldsymbol{b}_n} \mathcal{L}_{ours} = \mathcal{L}\big(\mathcal{M}(\boldsymbol{x}'), y_t\big) + \alpha \cdot \mathcal{L}_{\text{pixel}}, \tag{3}$$

$$\text{where } \boldsymbol{x}' = \text{clip}\Big(\boldsymbol{x} + \frac{1}{2}\big(\tanh(\boldsymbol{b}_p) + 1\big) \cdot maxp - \frac{1}{2}\big(\tanh(\boldsymbol{b}_n) + 1\big) \cdot maxp\Big), \tag{4}$$

$$\text{and } \mathcal{L}_{\text{pixel}} = \sum_{h,w}\Big(\max_c\Big(\frac{1}{2}\big(\tanh(\frac{\boldsymbol{b}_p}{\gamma}) + 1\big)\Big)\Big) + \sum_{h,w}\Big(\max_c\Big(\frac{1}{2}\big(\tanh(\frac{\boldsymbol{b}_n}{\gamma}) + 1\big)\Big)\Big). \tag{5}$$

Variables $\boldsymbol{b}_p,\ \boldsymbol{b}_n \in (-\infty, +\infty)$ denote positive and negative perturbations, respectively; $\mathcal{L}(\cdot,\cdot)$ denotes the cross entropy loss function of the subject model $\mathcal{M}$; y_t is the target label; α controls the weight of the second objective. We dynamically adjust α according to the attack success rate during optimization to better balance the two objectives. Operation $\text{clip}(\cdot)$ constrains the values to the valid pixel value range. In Equation 4, $\frac{1}{2}(\tanh(\boldsymbol{b}_p)+1)\cdot maxp$ denotes the positive value change and $\frac{1}{2}(\tanh(\boldsymbol{b}_n)+1)\cdot maxp$ the negative change, with $maxp$ the upper bound of pixel values (i.e., 255). The function $\sum_{h,w}$ sums perturbations at all pixels with $\max_c$ the maximum among the three R, G, B channels. Parameter γ is used to alter the slope of $\tanh$ such that the optimization is smoother. We empirically set $\gamma = 10$.

A Simplified Version. Empirically we find that when using $\tanh$ in perturbing pixel values (in Equation 4), the optimizer continues to have gradient descents from the cross-entropy loss term in Equation 3, which is much more complex than the $\mathcal{L}_{\text{pixel}}$ term, to variables $\boldsymbol{b}_p$ and $\boldsymbol{b}_n$, even when the pixel value changes (e.g., $\frac{1}{2}(\tanh(\boldsymbol{b}_p)+1)\cdot maxp$) are already close to 0. This unnecessarily slows down the optimization. We hence replace Equation 4 with the following.

$$\boldsymbol{x}' = \text{clip}\big(\boldsymbol{x} + \text{clip}(\boldsymbol{b}_p \cdot maxp) - \text{clip}(\boldsymbol{b}_n \cdot maxp)\big), \tag{6}$$

Specifically, we remove the $\tanh$ functions on $\boldsymbol{b}_p$ and $\boldsymbol{b}_n$. Instead, we directly scale them with $maxp$ and then clip them to the valid range. This is equivalent to using a linear function in the cross-entropy loss term in Equation 3 instead of $\tanh$, while keeping the $\tanh$ functions in the $\mathcal{L}_{\text{pixel}}$ loss term. Intuitively, the shape of $\text{clip}(\boldsymbol{b}_p \cdot maxp)$ is similar to that of a $\tanh$ function. That is, the values on the two sides are zero and maximum, and there is a slope within a small range in the center. As such, Equation 6 approximates Equation 4. Empirically, we find that it makes our method faster and does not degrade the quality of generated triggers when it is used to generate natural triggers. It is faster because the clip operations prevent unnecessary gradient descents. However, we also find that Equation 4 is necessary in generating injected triggers for trojaned models during backdoor scanning (see Section 5.5). We speculate trojaned models have more non-linear behaviors than clean models due to data poisoning, which requires a smoother loss function. Specifically, trojaned models need to learn not only the relations between normal features and correct labels, but also the relations between poisoned data and the target label. This requires them to have more complex decision boundaries than benign models, and hence more non-linear behaviors. Smoother functions help escaping local optima with the increased non-linearity of trojaned models. Moreover, our ablation study in Appendix K shows that the $\tanh$ in Equation 5 is always beneficial.

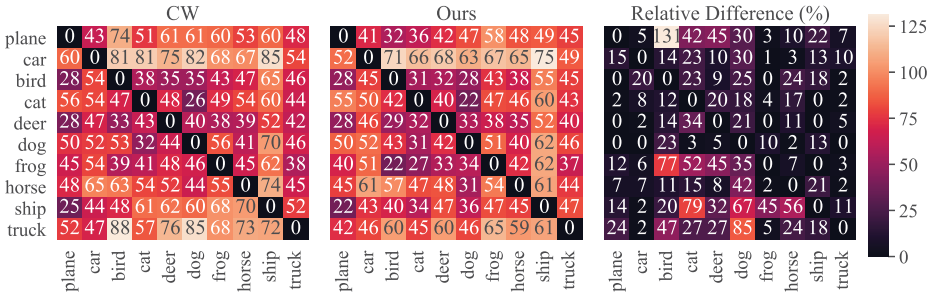
5. Evaluation

The evaluation is conducted on four datasets including ImageNet. For backdoor scanning, we leverage pre-trained models from the TrojAI competition [50] with a variety of classification tasks and model types. We also conduct an ablation study to understand the effects of different design choices (see Appendix K). Most experiments are conducted on a server equipped with two Intel Xeon Silver 4214 2.20GHz 12-core processors, 256 GB of RAM, and eight NVIDIA Quadro RTX 6000 GPUs.

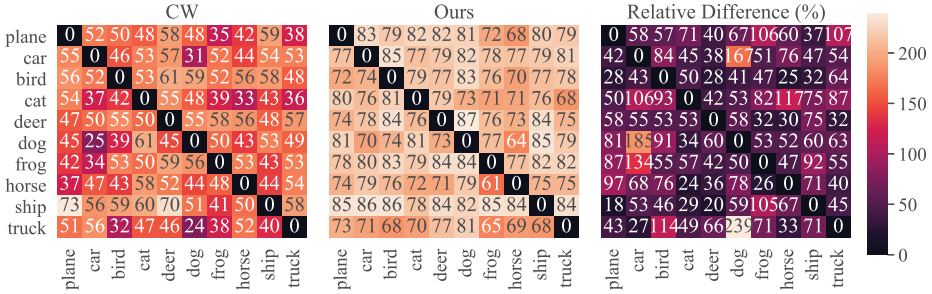
5.1. Experiment Setup

Datasets and Models. We use four datasets: CIFAR-10 [27], SVHN [47], LISA [45] and ImageNet [54]. We also conduct experiments on 300 pre-trained models (including clean and poisoned models) from rounds 2-4 of TrojAI competition [50]. Details are in Appendix C.

Baselines. Three existing optimization methods discussed in Section 3 are employed as the baselines: NC [65], CW [4], and UAP [58]. We randomly select 100 images from the validation set as the generation set for CIFAR-10 and SVHN, that is, the set of clean images used for trigger generation. For ImageNet, CW can only be performed on 50 images given the GPU memory limit. We hence randomly select 50 images from the training set as the generation set for all the methods. We use 90% ASR as the threshold on the generation set for CW, NC and ours. Since UAP may not produce any trigger with a high ASR, we do not use the threshold for UAP. As UAP is an L^∞ attack, we use an L^∞ bound of $8/255$ for CIFAR-10 and ImageNet, and 0.03 for SVHN. Due to the different natures of these methods, it is hard to define a uniform criterion (threshold) of convergence. For fair comparison, we use a conservative (i.e., fairly large) number of optimization epochs (1000



(a) #Pixels of generated triggers. The last heat map shows how much larger CW triggers are.



(b) ASRs of generated backdoors. The last map shows how much higher our ASRs are

Figure 5. Comparison of CW and ours for all class pairs on CIFAR-10

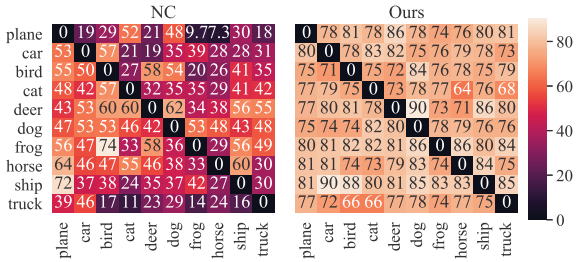


Figure 6. Comparison of NC and ours on the ASR for all class pairs on the CIFAR-10 dataset

epochs) for all the methods. Note that both CW and NC converge slower than ours. Please see the results on SVHN in Appendix F and the comparison with UAP in Appendix E due to the page limit.

Metrics. We consider the following criteria. The number of perturbed pixels (#pixels) measures the size of generated triggers. The attack success rate (ASR) gauges the percentage of unseen clean samples that can be flipped by a trigger. For evaluating ASR, we use the whole test set for CIFAR-10 and SVHN, and the whole validation set for ImageNet. We also measure the time cost.

5.2. Evaluation on CIFAR-10

Comparison with CW Optimization. In this experiment, we use CW and our method to generate natural triggers for all the class pairs for a clean ResNet20 model on CIFAR-10. Figure 5 shows the comparison. Each cell in heat map denotes the result for a natural backdoor flipping all the test samples from a victim class (row) to a target class (column). Figure 5a and Figure 5b show the number of perturbed pixels and the ASRs for CW (the left heat map) and ours (the middle heat map), respectively. The right heat map in Figure 5a shows how much larger the CW triggers are compared to ours. Observe that there are a few class pairs where CW and ours have the same trigger size, such as bird→plane and deer→plane. However, for other pairs, CW has a significantly larger trigger size than ours. For instance, for pair plane→bird, the trigger by CW is 131% larger than ours. Even with a much larger trigger, CW however still has lower ASR (50% vs 79% for plane→bird). This is because CW uses an external procedure to reduce the number of perturbed pixels (removing unimportant pixels based on $\boldsymbol{g}_i \cdot \boldsymbol{\delta}_i$ as discussed in Appendix A). Our method converges 10.88 times faster than CW on average (see Appendix D).

Comparison with NC. NC tends to generate triggers with a large number of small perturbations. The generated triggers hence cannot be easily applied in physical attacks. We conduct two experiments: (1) align the number of perturbed pixels of the NC triggers and our triggers and then compare the corresponding ASRs; (2) align the ASRs and compare the trigger sizes. For the first experiment, we use the sizes of our triggers as the reference, and align the NC triggers by gradually removing their smallest perturbations until they have the same sizes as ours. We then compare the ASRs of our triggers and the reduced NC triggers. Figure 6 presents the results. Observe that for most class pairs, the reduced NC triggers have less than 50% ASR. In the worst case, NC has only 7.3% ASR (plane→horse). On average, NC

Table 1. Comparison of different methods on a victim class logger-head turtle (left table) and a victim class Persian cat (right table) from ImageNet. The first column shows the target classes. The second column shows the methods. The third/sixth column is the time cost in minutes and the fourth/seventh column the number of perturbed pixels (#Pixels). The fifth/eighth column shows ASR on the samples from validation set.

T	Method	Time	#Pixels	ASR	Time	#Pixels	ASR
Snowbird	CW	845.57	1849	0.00%	850.49	1097	4.00%
	UAP	21.19	50171	0.00%	22.44	50175	10.00%
	NC	9.19	26032	60.00%	9.35	25887	58.00%
	Ours	4.35	432	72.00%	4.43	519	66.00%
Robin	CW	1039.72	1674	0.00%	983.07	1063	2.00%
	UAP	21.77	50172	0.00%	22.85	50176	14.00%
	NC	9.19	26094	34.00%	9.44	26358	46.00%
	Ours	4.10	467	60.00%	4.52	433	54.00%
Grouse	CW	1035.85	2150	0.00%	882.65	1340	2.00%
	UAP	22.94	50174	0.00%	22.10	50174	12.00%
	NC	9.52	25977	14.00%	9.10	25688	44.00%
	Ours	4.02	675	60.00%	4.35	656	54.00%
Kangaroo	CW	1079.54	2165	0.00%	1028.50	1503	0.00%
	UAP	22.69	50173	2.00%	22.52	50176	8.00%
	NC	9.02	26583	28.00%	9.10	29165	54.00%
	Ours	4.27	822	70.00%	4.35	621	62.00%

has 39.83% ASR for all class pairs, degraded from 65.52% without reduction. This demonstrates that the large number of perturbations in NC triggers are important for a good ASR although they may have small values. In contrast, our triggers have higher ASRs than NC's for all class pairs. On average, ours have 78.22% ASR, even higher than the original NC triggers without size reduction. In the second experiment, we use NC's ASR as the reference and then gradually remove the smallest perturbations in our triggers until their ASRs drop to the same level as NC's and then compare the sizes. Figure 10a in Appendix presents the results. Observe that NC has one order of magnitude larger trigger sizes than ours for all the class pairs, indicating that our generated triggers indeed perturb much fewer pixels. We also study an NC variant, ABS [38], for trigger generation and have similar observations in Appendix G.

5.3. Evaluation on ImageNet

ImageNet has 1,000 classes. It is hence infeasible to test on all class pairs, especially for CW, which takes more than 14 hours to generate just one trigger. We hence randomly select 8 class pairs for experiments (see results on more class pairs in Table 9 in Appendix). Table 1 presents the quality of generated triggers. Observe that CW takes more than 800 minutes to generate a trigger for all the evaluated class pairs, and the highest ASR it can achieve is 4% for pair cat→snowbird. The size of generated triggers by CW is smaller than UAP and NC, but one order of magnitude larger than ours. UAP is much faster than CW, but is still one order of magnitude slower than ours. Its ASRs are also very low, with the highest 14%. Compared to the other two baselines, NC is faster and has a better ASR (42.25% on average). However, the triggers by NC have more than 25k perturbed pixels, which are almost half of the whole image ($224 \times 224 \approx 50k$). Our method has the lowest time cost, requiring less than 5 minutes to generate a valid trigger with a higher ASR (62.26% on average). Compared to NC, our triggers are two orders of magnitude smaller and have 20% higher ASR. We also conduct an experiment similar to the above on a desktop to demonstrate that our method can be easily deployed on machines with limited resources (see Appendix H). We further study the robustness of generated triggers under various image transformations. Results show that most of NC triggers become ineffective after 96% rescaling or 2° rotation (nearly 0% ASR). Our method has a consistently higher ASR than NC (see details in Appendix I). We also study the robustness of triggers by applying transformations during trigger generation. The observations are similar (see Appendix J).

Table 2. Comparison of different methods on model hardening. First two columns denote different training methods and model accuracy. The third and the fifth columns show the average trigger size measured by NC and ours, respectively. The fourth and the sixth columns denote the improvement.

Method	Accuracy	Adv_{NC}	$\text{Increase}_{\text{NC}}$	Adv_{Ours}	$\text{Increase}_{\text{Ours}}$
Natural	95.15%	55.11	-	32.83	-
UAP	93.16%	49.40	-8.86%	23.69	-27.08%
NC	93.45%	75.77	39.10%	45.57	39.69%
Ours	94.18%	122.79	121.07%	83.24	152.02%

5.4. Model Hardening

As natural backdoors widely exist in clean models. It is important to harden models against such attacks. We use the generated triggers by different methods to harden models and then apply NC and our method to generate triggers for all class pairs to measure improvement. Table 2 shows the results on a ResNet32 model for SVHN. Observe that the improvement on average trigger size by our method is 3x larger than those by existing methods (i.e., UAP and NC). We evaluate on two more datasets and five more models, and the observation is similar. Please see details in Appendix L.1.

5.5. Backdoor Scanning

We study the performance of existing backdoor scanners by replacing their trigger inversion method with ours on the polygon attack and three advanced backdoor attacks.

For polygon backdoors, we evaluate on 300 pre-trained models from the TrojAI competition. The results show our method can improve a state-of-the-art scanner K-arm [59]'s accuracy by 2% via replacing its optimization component

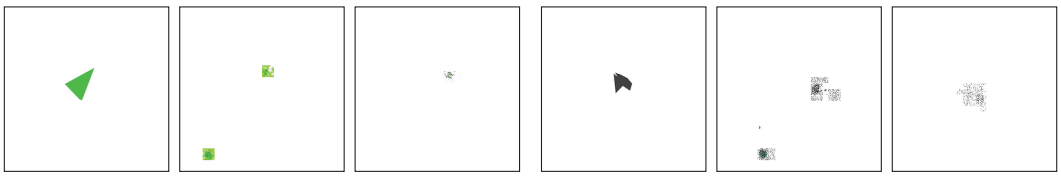


Figure 7. Comparison between injected backdoor and reverse engineered backdoor for poisoned models from the TrojAI dataset. Columns Injected show the original injected backdoors. Columns NC and Ours present backdoors generated by NC and ours, respectively. The numbers in the brackets denote the number of perturbed pixels of corresponding backdoors.

Table 3. Detecting a new pervasive backdoor attack [48]

Method	Dataset			
	MNIST	CIFAR-10	GTSRB	CelebA
NC	1.51	1.74	1.61	1.03
Ours	3.15	2.25	2.59	3.04

(based on NC) with ours. Note that the original scanner already had a state-of-the-art detection accuracy close to 90% such that 2% improvement is non-trivial. We also demonstrate example backdoors generated by NC and our method in Figure 7. The three images on the left show the injected trigger, the triggers inverted by NC and by ours, respectively. The three images on the right show another example. The number beside the method name denotes the trigger size. Observe that our generated backdoors are significantly smaller than NC's. Especially for the left case, ours is one order of magnitude smaller than that of NC. It is important to have small inverted triggers as scanners rely on the size of those triggers to distinguish poisoned models from benign ones. Note that in the TrojAI competition, the location of injected triggers is randomized to make the triggers more robust. The location shown in Figure 7 is only one of such cases. The generated triggers may be at any location.

We also evaluate our method on detecting three advanced backdoor attacks, namely, WaNet [48], invisible backdoor [32], and blind backdoor [2]. Compared to simple patch backdoors, WaNet and invisible backdoor have triggers that are not fixed. Their triggers are content based distortions. Blind backdoor uses inverted backdoors by existing scanners to adversarially train backdoored models, making the attack robust. We use the same anomaly index to detect backdoored models as that in the original NC paper, namely, a model with an anomaly index larger than 2 is considered backdoored. We download all the publicly available pre-trained models from WaNet [48]. Table 3 shows the anomaly indices for different models using NC and ours. We can see that NC cannot detect any of the evaluated models (consistent with the results reported in [48]), whereas our method can detect all the backdoored models (as we can generate a much smaller trigger for the target). The observation on the other two attacks are the same. Our inspection shows that although the injected triggers are pervasive, the models pick up low level features such as curly lines during poisoning. NC generates large triggers for the target class that are not distinguishable from those of benign classes, whereas our triggers are much smaller. Please see Appendix L.2 for more details. Our method is also consistently superior in detecting invisible backdoor and blind backdoor. Please see Appendix L.2.

6. Conclusion

We propose a new optimization method for backdoor trigger generation that minimizes the number of perturbed pixels. Compared to the state-of-the-art methods, our method is more cost-effective and can generate triggers with a smaller size, higher attack success rate, and better robustness. It also improves performance of model hardening and backdoor scanning.

Limitations of Our Method. Similar to NC and other existing scanners [19, 38, 59], our technique requires using a (small) set of clean samples in optimization, trying to flip their classification results. There are situations in which clean samples may not be available. It is unclear how our method can be extended to handle those cases. We will leave it to our future work.

Potential Negative Societal Impacts. The proposed method is general, aiming to generate better backdoor triggers. It could serve both attack and defense. Malicious users could use our method to generate triggers for pre-trained models and use them in attack. However, just like adversarial attack techniques are critical to improving model robustness, the triggers generated by our technique can be used to scan and mitigate backdoor vulnerabilities.

Acknowledgement

We thank the anonymous reviewers for their constructive comments. This research was supported, in part by IARPA TrojAI W911NF-19-S-0012, NSF 1901242 and 1910300, ONR N000141712045, N000141410468 and N000141712947. Any opinions, findings, and conclusions in this paper are those of the authors only and do not necessarily reflect the views of our sponsors.

References

[1] PixelBackdoor. https://github.com/Gwinhen/PixelBackdoor, 2022. 2

[2] Eugene Bagdasaryan and Vitaly Shmatikov. Blind backdoors in deep learning models. In *USENIX Security 21*, 2021. 8, 19

[3] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *AISTATS 2020*, pages 2938–2948. PMLR, 2020. 3

[4] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *SP 2017*, pages 39–57. IEEE, 2017. 2, 3, 4, 5

[5] Alvin Chan and Yew-Soon Ong. Poison as a cure: Detecting & neutralizing variable-sized backdoor attacks in deep neural networks. *arXiv preprint arXiv:1911.08040*, 2019. 3

[6] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728*, 2018. 3

[7] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017. 3

[8] Edward Chou, Florian Tramer, and Giancarlo Pellegrino. Sentinet: Detecting localized universal attack against deep learning systems. *SPW 2020*, 2020. 3

[9] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *CVPR 20*, pages 3213–3223, 2016. 13

[10] Min Du, Ruoxi Jia, and Dawn Song. Robust anomaly detection and backdoor attack detection via differential privacy. In *ICLR 19*, 2019. 3

[11] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *CVPR 18*, pages 1625–1634, 2018. 12, 13

[12] Hao Fu, Akshaj Kumar Veldanda, Prashanth Krishnamurthy, Siddharth Garg, and Farshad Khorrami. Detecting backdoors in neural networks using novel feature-based anomaly detection. *arXiv preprint arXiv:2011.02526*, 2020. 3

[13] Yansong Gao, Change Xu, Derui Wang, Shiping Chen, Damith C Ranasinghe, and Surya Nepal. Strip: A defence against trojan attacks on deep neural networks. In *ACSAC 19*, pages 113–125, 2019. 3

[14] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *IJRR*, 2013. 13

[15] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. *arXiv preprint arXiv:1412.6572*, 2014. 3

[16] Ian J Goodfellow, Oriol Vinyals, and Andrew M Saxe. Qualitatively characterizing neural network optimization problems. In *ICLR 2015*, 2015. 1

[17] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, 7:47230–47244, 2019. 1, 3

[18] Wenbo Guo, Lun Wang, Xinyu Xing, Min Du, and Dawn Song. Tabor: A highly accurate approach to inspecting and restoring trojan backdoors in ai systems. *arXiv preprint arXiv:1908.01763*, 2019. 18

[19] Wenbo Guo, Lun Wang, Yan Xu, Xinyu Xing, Min Du, and Dawn Song. Towards inspecting and eliminating trojan backdoors in deep neural networks. In *ICDM*, 2020. 1, 3, 8, 18

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 12, 13

[21] Shanjiaoyang Huang, Weiqi Peng, Zhiwei Jia, and Zhuowen Tu. One-pixel signature: Characterizing cnn models for backdoor detection. In *ECCV*, 2020. 1, 3, 18

[22] Xijie Huang, Moustafa Alzantot, and Mani Srivastava. Neuroninspect: Detecting backdoors in neural networks via output explanations. *arXiv preprint arXiv:1911.07399*, 2019. 1, 3, 18

[23] Jinyuan Jia, Xiaoyu Cao, and Neil Zhenqiang Gong. Intrinsic certified robustness of bagging against data poisoning attacks. *arXiv preprint arXiv:2008.04495*, 2020. 3

[24] Daniel Jiwoong Im, Michael Tao, and Kristin Branson. An empirical analysis of the optimization of deep network loss surfaces. *arXiv e-prints*, pages arXiv–1612, 2016. 1

[25] Keras. Applications. https://keras.io/api/applications/, 2021. 2, 13

[26] Soheil Kolouri, Aniruddha Saha, Hamed Pirsiavash, and Heiko Hoffmann. Universal litmus patterns: Revealing backdoor attacks in cnns. In *CVPR*, pages 301–310, 2020. 3, 18

[27] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 5, 12

[28] Keita Kurita, Paul Michel, and Graham Neubig. Weight poisoning attacks on pre-trained models. In *ACL 20*, 2020. 3

[29] Fredrik Larsson, Michael Felsberg, and P-E Forssen. Correlating fourier descriptors of local patches for road sign recognition. *IET Computer Vision*, 5(4):244–254, 2011. 13

[30] Alexander Levine and Soheil Feizi. Deep partition aggregation: Provable defense against general poisoning attacks. *arXiv preprint arXiv:2006.14768*, 2020. 3

[31] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 6391–6401, 2018. 1

[32] Shaofeng Li, Minhui Xue, Benjamin Zhao, Haojin Zhu, and Xinpeng Zhang. Invisible backdoor attacks on deep neural networks via steganography and regularization. *TDSC 20*, 2020. 8, 19

[33] Yige Li, Nodens Koren, Lingjuan Lyu, Xixiang Lyu, Bo Li, and Xingjun Ma. Neural attention distillation: Erasing backdoor triggers from deep neural networks. In *ICLR 21*, 2021. 3

[34] Yiming Li, Tongqing Zhai, Baoyuan Wu, Yong Jiang, Zhifeng Li, and Shutao Xia. Rethinking the trigger of backdoor attack. *arXiv preprint arXiv:2004.04692*, 2020. 3

[35] Junyu Lin, Lei Xu, Yingqi Liu, and Xiangyu Zhang. Composite backdoor attack for deep neural network by mixing existing benign features. In *CCS 20*, pages 113–131, 2020. 1, 3

[36] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013. 12

[37] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *RAID 18*, pages 273–294. Springer, 2018. 3

[38] Yingqi Liu, Wen-Chuan Lee, Guanhong Tao, Shiqing Ma, Yousra Aafer, and Xiangyu Zhang. Abs: Scanning neural networks for back-doors by artificial brain stimulation. In *CCS 19*, pages 1265–1282, 2019. 1, 2, 3, 7, 8, 15, 18

[39] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. In *NDSS 18*, 2018. 1

[40] Yunfei Liu, Xingjun Ma, James Bailey, and Feng Lu. Reflection backdoor: A natural backdoor attack on deep neural networks. In *ECCV 20*, pages 182–199. Springer, Cham, 2020. 1

[41] Yingqi Liu, Guangyu Shen, Guanhong Tao, Zhenting Wang, Shiqing Ma, and Xiangyu Zhang. Ex-ray: Distinguishing injected backdoor from natural features in neural networks by examining differential feature symmetry. *arXiv preprint arXiv:2103.08820*, 2021. 1, 3

[42] Yuntao Liu, Yang Xie, and Ankur Srivastava. Neural trojans. In *ICCD 17*, pages 45–48. IEEE, 2017. 3

[43] Shiqing Ma, Yingqi Liu, Guanhong Tao, Wen-Chuan Lee, and Xiangyu Zhang. Nic: Detecting adversarial samples with neural network invariant checking. In *NDSS 19*, 2019. 3

[44] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *ICLR 18*, 2018. 3, 13

[45] Andreas Mogelmose, Mohan Manubhai Trivedi, and Thomas B Moeslund. Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey. *T-ITS*, 13(4):1484–1497, 2012. 5, 12

[46] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *CVPR 17*, pages 1765–1773, 2017. 3, 4

[47] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011. 5, 12

[48] Anh Nguyen and Anh Tran. Wanet–imperceptible warping-based backdoor attack. In *ICLR 2021*, 2021. 8, 19

[49] Tuan Anh Nguyen and Anh Tran. Input-aware dynamic backdoor attack. *NeurIPS 20*, 2020. 3

[50] NIST. TrojAI. https://pages.nist.gov/trojai/, 2020. 5, 13, 18

[51] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *EuroS&P 16*, pages 372–387. IEEE, 2016. 3, 4

[52] Ximing Qiao, Yukun Yang, and Hai Li. Defending neural backdoors via generative distribution modeling. In *NeurIPS 19*, pages 14004–14013, 2019. 3, 18

[53] Shahbaz Rezaei and Xin Liu. A target-agnostic attack on deep models: Exploiting security vulnerabilities of transfer learning. In *ICLR*, 2020. 3

[54] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015. 5, 13

[55] Aniruddha Saha, Akshayvarun Subramanya, and Hamed Pirsiavash. Hidden trigger backdoor attacks. In *AAAI 20*, number 07, pages 11957–11965, 2020. 3

[56] Ahmed Salem, Rui Wen, Michael Backes, Shiqing Ma, and Yang Zhang. Dynamic backdoor attacks against machine learning models. *arXiv preprint arXiv:2003.03675*, 2020. 3

[57] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *NeurIPS 18*, pages 6103–6113, 2018. 3

[58] Ali Shafahi, Mahyar Najibi, Zheng Xu, John Dickerson, Larry S Davis, and Tom Goldstein. Universal adversarial training. In *AAAI 20*, volume 34, pages 5636–5643, 2020. 2, 4, 5, 12

[59] Guangyu Shen, Yingqi Liu, Guanhong Tao, Shengwei An, Qiuling Xu, Siyuan Cheng, Shiqing Ma, and Xiangyu Zhang. Backdoor scanning for deep neural networks through k-arm optimization. In *ICML*, 2021. 1, 2, 7, 8, 18

[60] Di Tang, XiaoFeng Wang, Haixu Tang, and Kehuan Zhang. Demon in the variant: Statistical analysis of dnns for robust backdoor contamination detection. In *USENIX Security*, 2021. 3

[61] Florian Tramèr and Dan Boneh. Adversarial training and robustness for multiple perturbations. In *NeurIPS 19*, 2019. 3

[62] Brandon Tran, Jerry Li, and Aleksander Madry. Spectral signatures in backdoor attacks. In *NeurIPS*, pages 8000–8010, 2018. 3

[63] Akshaj Kumar Veldanda, Kang Liu, Benjamin Tan, Prashanth Krishnamurthy, Farshad Khorrami, Ramesh Karri, Brendan Dolan-Gavitt, and Siddharth Garg. Nnoculation: broad spectrum and targeted treatment of backdoored dnns. *arXiv preprint arXiv:2002.08313*, 2020. 3, 18

[64] Binghui Wang, Xiaoyu Cao, Neil Zhenqiang Gong, et al. On certifying robustness against backdoor attacks via randomized smoothing. *arXiv preprint arXiv:2002.11750*, 2020. 3

[65] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *S&P 19*, pages 707–723. IEEE, 2019. 1, 2, 3, 5, 17, 18, 19

[66] Bolun Wang, Yuanshun Yao, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. With great training comes great

vulnerability: Practical attacks against transfer learning. In *USENIX Security 18*, pages 1281–1297, 2018. 3

[67] Hongyi Wang, Kartik Sreenivasan, Shashank Rajput, Harit Vishwakarma, Saurabh Agarwal, Jy-yong Sohn, Kangwook Lee, and Dimitris Papailiopoulos. Attack of the tails: Yes, you really can backdoor federated learning. *NeurIPS*, 33, 2020. 3

[68] Ren Wang, Gaoyuan Zhang, Sijia Liu, Pin-Yu Chen, Jinjun Xiong, and Meng Wang. Practical detection of trojan neural networks: Data-limited and data-free cases. In *ECCV*, 2020. 1, 3, 18

[69] Maurice Weber, Xiaojun Xu, Bojan Karlas, Ce Zhang, and Bo Li. Rab: Provable robustness against backdoor attacks. *arXiv preprint arXiv:2003.08904*, 2020. 3

[70] Eric Wong, Leslie Rice, and J Zico Kolter. Fast is better than free: Revisiting adversarial training. In *ICLR*, 2020. 13

[71] Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. Dba: Distributed backdoor attacks against federated learning. In *ICLR*, 2019. 3

[72] Xiaojun Xu, Qi Wang, Huichen Li, Nikita Borisov, Carl A Gunter, and Bo Li. Detecting ai trojans using meta neural analysis. *arXiv preprint arXiv:1910.03137*, 2019. 3, 18

[73] Yuanshun Yao, Huiying Li, Haitao Zheng, and Ben Y Zhao. Latent backdoor attacks on deep neural networks. In *CCS*, pages 2041–2055, 2019. 3

[74] Yi Zeng, Han Qiu, Shangwei Guo, Tianwei Zhang, Meikang Qiu, and Bhavani Thuraisingham. Deepsweep: An evaluation framework for mitigating dnn backdoor attacks using data augmentation. *arXiv preprint arXiv:2012.07006*, 2020. 3

[75] Xinqiao Zhang, Huili Chen, and Farinaz Koushanfar. Tad: Trigger approximation based black-box trojan detection for ai. *arXiv preprint arXiv:2102.01815*, 2021. 1

[76] Xiaoyu Zhang, Ajmal Mian, Rohit Gupta, Nazanin Rahnavard, and Mubarak Shah. Cassandra: Detecting trojaned networks from adversarial perturbations. *arXiv preprint arXiv:2007.14433*, 2020. 1, 3, 18

[77] Xinyang Zhang, Zheng Zhang, and Ting Wang. Trojaning language models for fun and profit. In *EuroS&P*, 2021. 3

[78] Shihao Zhao, Xingjun Ma, Xiang Zheng, James Bailey, Jingjing Chen, and Yu-Gang Jiang. Clean-label backdoor attacks on video recognition models. In *CVPR 20*, pages 14443–14452, 2020. 3

[79] Liuwan Zhu, Rui Ning, Cong Wang, Chunsheng Xin, and Hongyi Wu. Gangsweep: Sweep out neural backdoors by gan. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 3173–3181, 2020. 18

Appendix

A. Optimization of CW

The CW L^0 attack first searches for perturbations on all pixels that can cause misclassification using the L^2 norm. It then uses a processing step external to the optimization to remove the perturbations that are the least important after each optimization epoch. The objective of optimization is the following.

$$\min_{\boldsymbol{w}} \mathcal{L}_{\text{CW}} = \|\boldsymbol{x}' - \boldsymbol{x}\|_2^2 + c \cdot f(\boldsymbol{x}'), \tag{7}$$

$$\text{where } f(\boldsymbol{x}') = \max\big(\max\{Z(\boldsymbol{x}')_i : i \neq t\} - Z(\boldsymbol{x}')_t, -\kappa\big), \tag{8}$$

$$\text{and } \boldsymbol{x}' = \boldsymbol{v} \circ \frac{1}{2}\tanh(\boldsymbol{w}) + (1 - \boldsymbol{v}) \circ \boldsymbol{x}. \tag{9}$$

The first objective in $\mathcal{L}_{\text{CW}}$ is to minimize the L^2 distance between the perturbed input $\boldsymbol{x}'$ and the original input $\boldsymbol{x}$. The second objective $f(\boldsymbol{x}')$ is for inducing misclassification by enlarging the target label t's logits value (denoted by $Z(\cdot)$) and reducing the largest logits of other labels as shown in Equation 8. The parameter κ controls the confidence of the misclassification. Equation 9 explains how an input $\boldsymbol{x}$ is perturbed. The perturbation is controlled by a mask vector $\boldsymbol{v}$ similar to the mask in NC. Its value is either 0 or 1, with the former indicating the corresponding pixel cannot be perturbed and the latter meaning that the pixel is replaced with the perturbation value. Variable $\boldsymbol{w}$ is a vector of arbitrary values denoting the perturbations. Function $\frac{1}{2}\tanh(\boldsymbol{w})$ projects these values to $[-0.5, 0.5]$. Note that in CW, each pixel value is normalized to $[-0.5, 0.5]$. To reduce the number of perturbed pixels, CW leverages a processing step external to the optimization after each epoch. Specifically, let $\boldsymbol{\delta} = \frac{1}{2}\tanh(\boldsymbol{w})$ be the perturbations and $\boldsymbol{g} = \nabla\mathcal{L}_{\text{CW}}$ be the gradient of objective function. It computes the importance of the perturbation of a pixel i by $\boldsymbol{g}_i \cdot \boldsymbol{\delta}_i$. CW sets $\boldsymbol{v}_i$ to 0 if the importance is smaller than a threshold. The change of $\boldsymbol{v}$ is monotonic. The algorithm can be easily adapted to generate backdoor: instead of optimizing $\boldsymbol{w}$ for one input, we optimize it for a set of inputs.

B. Optimization of Universal Adversarial Perturbation (UAP)

The existing UAP algorithm mainly produces perturbations for *untargeted* attacks. Since backdoor attacks are usually targeted, we extend the algorithm as follows.

$$\min_{\boldsymbol{\delta}} \mathcal{L}_{\text{UAP}} = \frac{1}{N}\sum_{i=1}^{N} \hat{\mathcal{L}}\big(\mathcal{M}(\boldsymbol{x}_i + \boldsymbol{\delta}), y_t\big) \text{ s.t. } \|\boldsymbol{\delta}\|_\infty \leq \epsilon, \tag{10}$$

$$\text{where } \hat{\mathcal{L}}\big(\mathcal{M}(\boldsymbol{x}_i + \boldsymbol{\delta}), y_t\big) = \min\{\mathcal{L}(\boldsymbol{x}_i + \boldsymbol{\delta}, y_t), \beta\}. \tag{11}$$

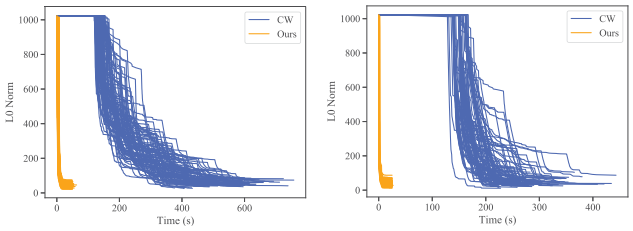
Variable $\boldsymbol{\delta}$ denotes the backdoor perturbation, bounded by an L^∞ size ϵ. Parameter β is a threshold for the loss of individual samples so as to avoid the loss of a single sample dominating the whole objective and to achieve a higher ASR [58]. Intuitively, the method aims to bound the maximum perturbation on a single pixel while minimizing the adversarial loss. Such bound is needed. Otherwise with an unbounded L^∞ distance, the optimization might completely change the input image in order to achieve its goal. We do not add a loss term to minimize the L^∞ distance because its ASR is low to begin with. Our results in Appendix E and F show that UAP cannot achieve high ASRs. Furthermore, almost all the pixels on an input image are perturbed. This makes its application in the physical world very difficult. The second and third images in the first row in Figure 2 (see Section 1) present the generated backdoors by UAP and its reduced version, respectively. Observe that almost all the pixels ($50k \approx 224 \times 224$) are perturbed while the ASR is only 2%. Reducing the number of perturbed pixels leads to 0% ASR. Its performance on CIFAR-10 is better. The green line in Figure 3 (see Section 3) shows that the ASR for a UAP trigger can be as high as 0.6 (the right end of the line). However, applying only the top 200 perturbations of UAP backdoor has nearly zero ASR, indicating most perturbations in the backdoor are important.

C. Detailed Experiment Setup

CIFAR-10 [27] is an object recognition dataset for a 10-class classification task, which contains 60,000 images. We split the whole dataset into three sets: 48,000 images for training, 2,000 for validation and 10,000 for testing. Two different models are utilized for this dataset: ResNet20 [20], Network in Network (NiN) [36].

SVHN (Street View House Numbers) [47] dataset contains house number digits extracted from Google Street View images, which consist of 73,257 training images and 26,032 test images. We further split the original training set into 67,257 samples for training and 6,000 samples for validation. We employ two models, NiN [36] and ResNet32 [20].

LISA [45] is a U.S. traffic sign dataset that contains 47 different road signs. However, the number of samples of different classes is not well-balanced, with some classes having very few images. We use the same setting as in an existing work [11] by choosing 18 most common classes based



(a) CIFAR-10 (b) SVHN

Figure 8. Comparison of CW and ours on the generation efficiency for all class pairs on CIFAR-10 and SVHN datasets. The x-axis denotes the time in seconds and the y-axis shows the size of backdoors during the trigger generation.

Figure 9. Comparison of UAP and ours on the ASR for all class pairs on the CIFAR-10 dataset

on the number of training examples, and split the dataset into 5,635 training samples, 704 validation samples and 704 test samples. We use two model structures for this dataset: A CNN model [11] that consists of three convolutional layers and one fully-connected layer, and a ResNet20 [20].

ImageNet [54] dataset is a large set for image classification with 1,000 labels, which contains 1,281,167 training images and 50,000 validation images. We use a ResNet50 [20] model downloaded from a widely-used model repository [25].

For backdoor scanning, we randomly select 100 models (half clean and half poisoned) in each round from rounds 2-4 of the TrojAI competition [50]. We exclude the round 1 models due to its simple poisoning settings. In total, we have 150 clean models and 150 poisoned models. The task of backdoor scanning is to determine whether a given model is poisoned or not. TrojAI models utilize 16 different structures such as DenseNet121, InceptionV3, MobileNetV2, etc. Each model is trained to classify synthetic traffic signs to between 5 and 45 classes. Input images are created by compositing a foreground object, e.g., a synthetic traffic sign, with a random background image from five different datasets in three categories from the KITTI dataset [14], the Cityscapes dataset [9] and the Swedish Roads dataset [29]. The organizer provides 2-50 clean images per class for each model in different rounds. Poisoned models are trojaned with various kinds of backdoors, including universal, label-specific and position-specific. We consider polygon triggers in this paper which are pixel patterns (e.g., polygons with solid color). Random transformations, such as shifting, titling, lighting, blurring, and weather effects, are applied during training to improve dataset diversity. Also, adversarial training with PGD (Projected Gradient Descent) [44] and FBF (Fast is Better than Free) [70] is leveraged to improve model quality in rounds 3-4.

D. Comparison of Time Cost with CW

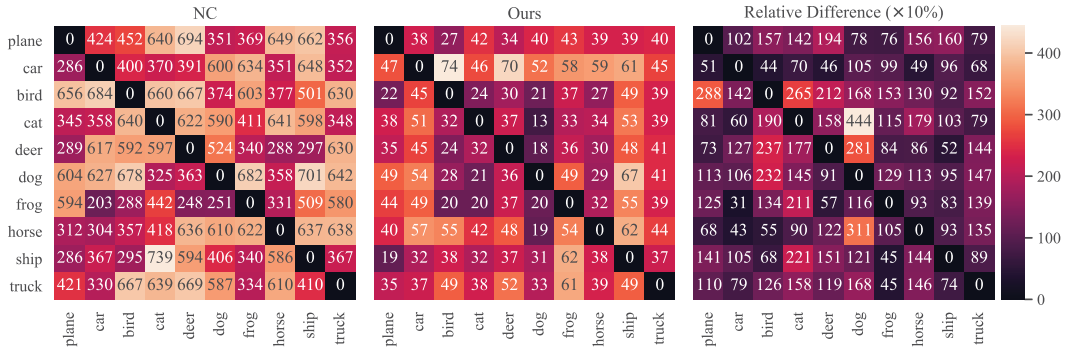
We compare the time cost of CW and ours in Figure 8. The x-axis denotes the time in seconds and the y-axis denotes the trigger size. We record the generation for all class pairs from CIFAR-10 and SVHN. Observe that CW spends a large amount of time finding a feasible solution at the beginning (around 120 seconds). After that, it aims to reduce the number of perturbed pixels using an external step as discussed in Section 3.2. This leads to the staircase phenomenon in Figure 8 as CW tries to find a feasible solution with the given set of pixels allowed for perturbing. Our method converges significantly faster than CW. On average, ours is 10.88 times faster on CIFAR-10 and 11.53 times faster on SVHN.

E. Comparison with UAP on CIFAR-10

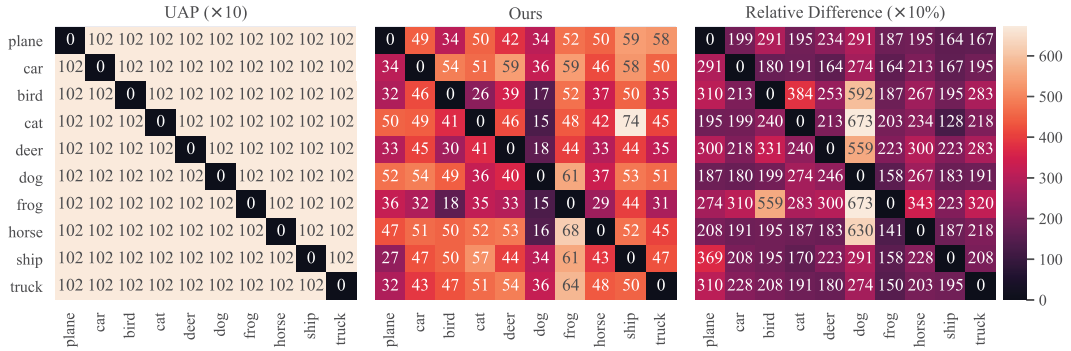
UAP is based on L^∞ and hence not directly comparable with our method. We follow the same procedure as before (see comparison with NC in the evaluation section): aligning by trigger sizes and then comparing ASRs, and aligning by ASRs and then comparing trigger sizes. The left heat map in Figure 9 shows the results when aligning trigger sizes. The average ASR of reduced UAP triggers is only 0.96%, whereas the original triggers have 68.21% ASR. It is clear that triggers generated by UAP are ineffective with a small number of perturbed pixels like ours. The left heat map in Figure 10b shows the results of aligning ASRs. Observe that the UAP triggers perturb all pixels, whereas ours are two orders of magnitude smaller. This is expected as UAP is an L^∞ method. Triggers generated by such a method can hardly be used in physical attack.

F. Evaluation on SVHN

Comparison with CW Optimization. In this experiment, we use CW and our method to generate natural triggers for all the class pairs for a clean NiN model on SVHN. Figure 12 shows the comparison. Each cell in a heat map denotes the result for a natural backdoor flipping all the test samples from a victim class (row) to a target class (column). Figure 12a and Figure 12b show the trigger sizes and the ASRs for CW (the left heat map) and ours (the middle



(a) Comparison with NC



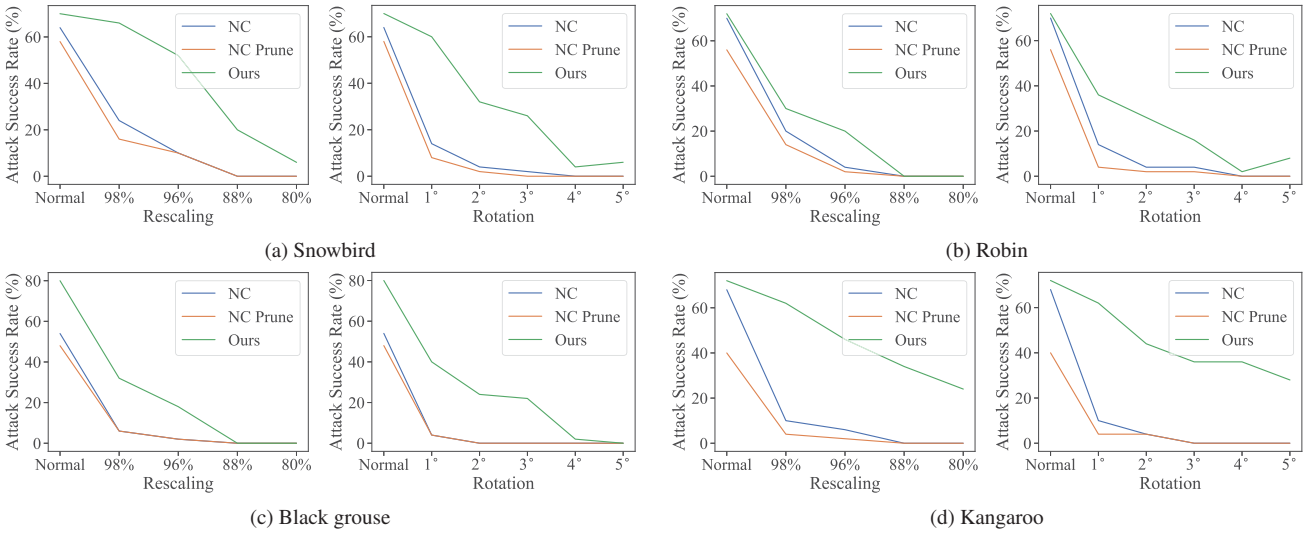
(b) Comparison with UAP

Figure 10. Comparison of the number of perturbed pixels with the same ASR for all class pairs on the CIFAR-10 dataset. The first two heat maps in each subfigure illustrate the results for NC/UAP and ours, respectively. The last heat map shows how much larger of generated backdoors by NC/UAP compared to ours.

heat map), respectively. The right heat map in Figure 12a shows how much larger the CW triggers are compared to ours. Observe that there are a few class pairs where CW and ours have the same trigger size, such as $3 \rightarrow 0$ and $5 \rightarrow 2$. However, for other pairs, CW has a significantly larger trigger size than ours. For instance, for pair $2 \rightarrow 0$, the trigger by CW is 110% larger than ours. Even with a much larger trigger, CW however still has lower ASR (59% vs 83% for $2 \rightarrow 0$). This is because CW uses an external procedure to reduce the number of perturbed pixels (removing unimportant pixels based on $\boldsymbol{g}_i \cdot \boldsymbol{\delta}_i$ as discussed in Section 3.2).

Comparison with NC. NC tends to generate triggers with a large number of small perturbations. The generated triggers hence cannot be easily applied in physical attacks. We conduct two experiments: (1) align the number of perturbed pixels of the NC triggers and our triggers and then compare the corresponding ASRs; (2) align the ASRs and compare the trigger sizes. For the first experiment, we use the sizes of our triggers as the reference, and align the triggers by NC by gradually removing their smallest perturbations until they have the same sizes as ours. We then compare the ASRs of our triggers and the reduced NC triggers. Figure 13 presents the results. Observe that for most class pairs, the reduced NC triggers have reasonable ASRs with an average of 75.87%, degraded from 76.55% without reduction. The results on SVHN are better than those on CIFAR-10 and ImageNet. Because SVHN is a dataset for digital number (from 0 to 9) recognition, which is a simpler task than objection recognition in CIFRA-10 and ImageNet. Optimization methods like NC can generate a digit shape (e.g., 1) with a few pixels change. This also explains the similar ASRs of triggers with and without reduction. Our triggers have higher ASRs than NC's for all class pairs, with the largest difference of 21% for $2 \rightarrow 8$ and $8 \rightarrow 2$. On average, ours have 83.18% ASR, even higher than the original NC triggers without size reduction. In the second experiment, we use NC's ASR as the reference and then gradually remove the smallest perturbations in our triggers until their ASRs drop to the same level as NC's and then compare the sizes. Figure 14a presents the results. Observe that NC has one order of magnitude larger trigger sizes than ours for all class pairs (except for $6 \rightarrow 5$ and $3 \rightarrow 8$). Since the reduced NC triggers have similar ASRs as the non-reduced ones, many pixel perturbations by NC are redundant and can be pruned for SVHN. After alignment, our generated triggers perturb fewer pixels.

Comparison with UAP. UAP is based on L^∞ and hence not directly comparable either. We follow the same proce-



(a) Snowbird (b) Robin

(c) Black grouse (d) Kangaroo

Figure 11. Evaluation of NC and our triggers under transformations on ImageNet. The victim class is turtle and the target classes are in the caption of each subfigure.

dure as before: aligning by trigger sizes and then comparing ASRs, and aligning by ASRs and then comparing trigger sizes. The middle heat map in Figure 13 shows the results when aligning trigger sizes. The average ASR of reduced UAP triggers is only 0.51%, whereas the original triggers have 18.76% ASR, which is also very low. It is clear that triggers generated by UAP are completely ineffective with a small number of perturbed pixels like ours. The left heat map in Figure 14b shows the results of aligning ASRs. Observe that the UAP triggers perturb all pixels, whereas ours are two orders of magnitude smaller.

G. Comparison with NC Variants

NC variants are based on NC and hence limited by the performance of NC. We test on a NC variant, ABS [38], for generating a trigger for pair plane→car on a ResNet20 model on CIFAR-10. The generated trigger by ABS has 84 perturbed pixels, which is twice larger than ours (38). The ASR on the test set is 38% by ABS and 78% by ours. If we reduce the number of perturbed pixels by ABS to match the size of our trigger, it has only 16% ASR. This empirically demonstrates that NC variants have the same limitation as NC. Besides, as those variants are built upon NC, they can be modified to use our method as the core optimization, which can boost their performance.

H. Evaluation on Desktop

We conduct an experiment on a desktop equipped with one Intel i7-8700 Processor, 16 GB of RAM, and a single NVIDIA GeForce GTX 1070 Ti GPU, which is a common affordable desktop machine. We use the case shown in Figure 2 in Section 1 and the results are shown in Table 4. Observe that the runtime for different methods on the desktop is similar to that on the server with minor differences. Our method still has the lowest time cost, and is around 2 times faster than NC and 4 times faster than UAP. Our method can be easily deployed on machines with limited resources.

Table 4. Experimental comparison on different machines

Method	Machine	Time	#Pixels	ASR
UAP	Server	22.69	50173	2.00%
	Desktop	21.12	50170	2.00%
NC	Server	9.02	26583	28.00%
	Desktop	10.62	17516	20.00%
Ours	Server	4.27	822	70.00%
	Desktop	5.40	818	70.00%

I. Robustness of Generated Triggers

We study the robustness of generated triggers under various image transformations. As the triggers by CW and UAP have very low ASRs, we hence only consider NC in this study. For some target classes, NC still has a low ASR. We then increase the strength of the adversary by utilizing 100 images for both NC and ours during trigger generation (50 images for the study in Section 5.3). This yields better test ASRs in general. We test on two types of transformations with different parameters: rescaling and rotation. Figure 11 presents the results for a source class turtle, with the target classes in individual subfigures. For each target class, we show the results under rescaling transformation on the left and under rotation on the right. The x-axis de-

Table 5. Comparison of different methods augmented with transformations during generation on a victim class turtle from the ImageNet dataset. The first column shows the target classes. The second column shows backdoor generation methods. The third column is the ASR of original triggers. The 4th-7th columns show the ASR under different rescaling transformations. The 8th-12th columns show the ASR under different rotation transformations. The last column shows the average ASR.

Target	Method	Normal	Rescaling				Rotation					Average
			98%	96%	88%	80%	1°	2°	3°	4°	5°	
Snowbird	NC	80.00%	74.00%	46.00%	0.00%	0.00%	82.00%	46.00%	8.00%	8.00%	2.00%	34.60%
	NC Prune	64.00%	44.00%	16.00%	0.00%	0.00%	50.00%	18.00%	0.00%	0.00%	0.00%	14.22%
	Ours	78.00%	66.00%	64.00%	16.00%	6.00%	74.00%	26.00%	18.00%	4.00%	4.00%	35.60%
Robin	NC	80.00%	70.00%	56.00%	16.00%	12.00%	68.00%	50.00%	20.00%	18.00%	10.00%	40.00%
	NC Prune	42.00%	38.00%	22.00%	6.00%	4.00%	32.00%	18.00%	4.00%	6.00%	4.00%	17.60%
	Ours	84.00%	78.00%	70.00%	46.00%	32.00%	80.00%	46.00%	40.00%	20.00%	12.00%	50.80%
Grouse	NC	82.00%	70.00%	48.00%	0.00%	0.00%	70.00%	6.00%	2.00%	0.00%	0.00%	27.80%
	NC Prune	76.00%	68.00%	42.00%	0.00%	0.00%	54.00%	4.00%	2.00%	0.00%	0.00%	18.89%
	Ours	82.00%	70.00%	68.00%	30.00%	28.00%	78.00%	42.00%	28.00%	24.00%	24.00%	47.40%
Kangaroo	NC	80.00%	72.00%	52.00%	0.00%	0.00%	74.00%	38.00%	34.00%	28.00%	20.00%	39.80%
	NC Prune	66.00%	46.00%	28.00%	0.00%	0.00%	52.00%	24.00%	16.00%	18.00%	8.00%	21.33%
	Ours	78.00%	70.00%	60.00%	50.00%	34.00%	76.00%	54.00%	52.00%	48.00%	40.00%	56.20%

notes the transformation strengths with the first one without any transformations, and the y-axis denotes the ASR. As we discussed in Section 5.3, NC has a much larger number of perturbed pixels than ours. Hence besides the original NC triggers, we also reduce the NC triggers to match our numbers of perturbed pixels and study their robustness as well. They are denoted as NC Prune in the figure. Observe that most of NC triggers become ineffective after 96% rescaling or 2° rotation (near 0% ASR). NC Prune has a lower ASR, even without transformations. For the target kangaroo, NC Prune has 28% lower ASR compared to NC, indicting that NC does require a large number of perturbed pixels to be effective, which is not so desirable for physical attacks. Our method has a consistently higher ASR than NC. For target kangaroo, our trigger has around 40% ASR with most of the transformations. We further study the robustness of triggers by applying transformations during the trigger generation. The observations are similar (see the following section).

J. Augmentation during Backdoor Generation

In this section, we study the robustness of triggers by applying transformations during trigger generation. Specifically, for input samples stamped with triggers, we randomly rescale 1% and rotate 1° for those samples. Using larger transformations would increase the trigger size, which is not desired for physical attacks. Table 5 shows the ASR results for augmented triggers on a victim class turtle from the ImageNet dataset. The first two columns show the target classes and backdoor generation methods. The third column is the ASR of original triggers. The 4th-7th columns show the ASR under different rescaling transformations. The 8th-12th columns show the ASR under different rotation transformations. The last column shows the average ASR. As NC has a much larger number of perturbed pixels than ours. Hence besides the original NC triggers, we also reduce the NC triggers to match our trigger sizes and study their robustness as well, which is denoted as NC Prune in the table. Observe that the ASRs of both NC and ours are increased on small scale transformations (98%-96% rescaling and 1°-2° rotation) compared to the results in Figure 11 in the previous section. However, for larger transformations, NC still has near 0% ASR on most cases. NC Prune has a low ASR even with the augmentation during the trigger generation. On average, it has only 18.01% ASR. Backdoors generated by our method can maintain a reasonable ASR. For instance, under the largest rescaling transformation (80%), our triggers still have around 30% ASR on the bottom three target classes. On average, our method has 47.50% ASR, 11.95% higher than NC (35.55%) and 29.49% higher than NC Prune (18.01%), indicating our generated triggers are more suitable for physical attacks.

K. Ablation Study

Our method introduces two components for approximating the number of perturbed pixels: the $\tanh$ loss and the two variables $\boldsymbol{b}_p$ and $\boldsymbol{b}_n$ for the positive and negative perturbations. We study individual components to understand their effects. In particular, we consider four settings, namely, (1) excluding the $\tanh$ loss; (2) replacing the two variables with a single variable for the positive perturbation; (3) replacing the two variables with a single variable for both positive and negative perturbations; (4) excluding both the $\tanh$ loss and the two variables. The results for pair plane→dog from CIFAR-10 on a ResNet20 model with 10 random runs are shown in Table 6. The number of perturbed pixels (#Pixels) is presented in the second column

Table 6. Ablation study on effects of different components. The results are collected from 10 random runs for the class pair plane→dog for a ResNet20 model on the CIFAR-10 dataset.

Method	#Pixels	ASR
Ours	38.70± 5.10	80.07±3.96%
- tanh loss	46.40±10.07	79.73±2.11%
- two variables (positive)	84.50±12.24	72.64±3.82%
- two variables	1024.00± 0.00	90.99±4.40%
- tanh loss & two variables	1023.10± 0.88	88.39±4.85%

and the ASR in the third column. Observe that without using the tanh loss, the trigger size increases by 20% from 38.70 to 46.40 on average. The standard deviation is twice of ours (10.07 vs. 5.10). Replacing the two variables with a single positive variable doubles the size of the generated trigger (from 38.70 to 84.50), and the ASR also drops (from 80.07% to 72.64%). Using a single variable for both positive and negative perturbations cannot reduce the number of perturbed pixels (the last two rows). As discussed in Section 4, there is a steep slope where the perturbation is 0 when using such a variable. It is unlikely for the optimization to stabilize at the 0 point. Hence, almost all the pixels will be perturbed during the trigger generation. This indicates the necessity of separating the positive and negative perturbations during the optimization. Using the tanh loss can further reduce the trigger size without sacrificing the ASR.

L. Applications

In this section, we evaluate our method in two applications including model hardening and backdoor scanning.

L.1. Model Hardening

In the introduction section, we have shown that a small backdoor generated by our method can flip the majority of samples of turtle to kangaroo in the ImageNet dataset. This is a critical security threat. We hence utilize generated triggers to harden the model by training on normal inputs stamped with triggers. After hardening, an adversary is supposed to produce a large and visible backdoor, which can be easily detectable by automated tools or human inspectors. CW is extremely expensive in trigger generation (as shown in Section 5 and Appendix D), which is not computationally feasible for model hardening that requires a large number of triggers generated on-the-fly during training. We hence only consider UAP, NC and ours for model hardening. In the original paper [65], NC generates universal backdoors for all classes beforehand, and then hardens the model using this set of backdoors by stamping on training inputs. We further improve the hardening process by generating universal backdoors on-the-fly, similar to adversarial training. We call it iterative NC. We use the same procedure for UAP and ours to harden models. The L^∞ bound for UAP training is determined according to the normal accuracy drop. We use L^∞ bound of $4/255$ for CIFAR-10, 0.05 for SVHN, and 0.03 for LISA.

We use the mask size by NC and the number of perturbed pixels by our method to measure the class distance from a victim class to a target class. The goal of model hardening is hence to enlarge the class distance for all pairs. We use the relative improvement of pairwise class distance as the metric. That is, we compute the improvement percentage for every class pair and obtain the average, defined as follows.

$$\frac{1}{n \times (n-1)} \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} \frac{\hat{d}_{i \to j} - d_{i \to j}}{d_{i \to j}}, \tag{12}$$

where n is the number of classes; $d_{i \to j}$ and $\hat{d}_{i \to j}$ are the class distances from i to j for the original model and the hardened model, respectively. We randomly select 100 samples from the validation set of class i and apply NC/ours for 1,000 epochs to generate a backdoor that can flip 90% of those samples to the target class j. As a backdoor is randomly initialized during generation, to avoid the bias from randomness, we run the generation on the same pair for 3 times and use the smallest backdoor size as the class distance. We show the average relative improvement along with the average class distance in the following results.

Table 7 shows the results for model hardening on CIFAR-10, SVHN and LISA datasets. The first three columns denote the dataset, model structure and training methods, respectively. The 4th column shows the model accuracy on the test set. The 5th column presents the training time in minutes. The 6th and 8th columns show the average class distance measured by NC and ours, respectively. The 7th and 9th columns show the relative improvement of class distance measured using Equation 12. UAP has the lowest improvement on distances by both NC and ours on CIFAR-10 and SVHN, except for the ResNet20 model on CIFAR-10 measured by NC. For cases such as ResNet32 on SVHN, UAP instead reduces the class distance measured by both NC and ours. NC can achieve a reasonable improvement from 4.58% to 39.10% measured by NC and from 3.96% to 49.56% measured by ours. The iterative version of NC further improves the class distance. It has an average of 68.61% improvement on the distance measured by NC and 89.70% by ours. Using our method for hardening produces the largest class distance improvement on both metrics (73.02% by NC and 106.37% by ours) on average. Specifically, models hardened by our method have much more improvements on the distance measured by ours, and are also better when measured by NC. This demonstrates that our backdoor generation method is better than NC in exposing model vulnerabilities. Models hard-

Table 7. Comparison of different methods on model hardening. First three columns denote different datasets (D), models (M) and training methods for the evaluation. The fourth column denotes model accuracy on the test set. The fifth column shows the training time in minutes. The sixth and the eighth columns show the average class distance across all class pairs measured by NC and ours, respectively. The seventh and the ninth columns denote the improvement of pairwise class distance (measured by NC and ours) by different techniques compared to that of original models (Natural).

D	M	Method	Accuracy	Time (m)	Adv_{NC}	$\text{Increase}_{\text{NC}}$	Adv_{Ours}	$\text{Increase}_{\text{Ours}}$
CIFAR-10	ResNet20	Natural	91.52%	56.77	53.49	-	43.91	-
		UAP	90.04%	243.11	96.00	81.57%	61.62	42.91%
		NC	90.83%	84.88	72.56	37.84%	62.77	49.56%
		Iterative NC	90.57%	65.00	93.54	78.16%	89.24	112.12%
		Ours	90.32%	64.11	95.17	79.21%	100.72	139.77%
	NiN	Natural	88.09%	68.30	60.67	-	38.17	-
		UAP	86.61%	196.67	57.56	-5.22%	36.92	-2.05%
		NC	86.64%	40.35	75.49	26.49%	52.20	42.89%
		Iterative NC	86.76%	33.90	90.69	54.09%	66.69	87.75%
		Ours	86.32%	28.46	93.77	59.08%	74.31	108.59%
SVHN	NiN	Natural	95.61%	10.50	64.63	-	37.17	-
		UAP	94.63%	45.47	69.31	6.99%	41.56	12.37%
		NC	94.89%	24.72	82.90	32.19%	53.16	48.96%
		Iterative NC	95.03%	53.40	107.15	65.97%	67.53	88.99%
		Ours	94.86%	42.71	108.43	68.47%	71.18	99.86%
	ResNet32	Natural	95.15%	26.70	55.11	-	32.83	-
		UAP	93.16%	228.95	49.40	-8.86%	23.69	-27.08%
		NC	93.45%	31.51	75.77	39.10%	45.57	39.69%
		Iterative NC	94.60%	109.30	120.20	113.92%	76.20	128.26%
		Ours	94.18%	97.55	122.79	121.07%	83.24	152.02%
LISA	CNN	Natural	97.30%	0.15	68.47	-	32.92	-
		UAP	95.60%	1.79	65.90	-1.23%	32.31	-0.43%
		NC	96.88%	8.27	71.69	4.58%	35.09	6.70%
		Iterative NC	96.45%	11.34	101.48	46.13%	53.38	60.36%
		Ours	96.02%	10.41	107.34	54.34%	56.83	70.11%
	ResNet20	Natural	98.86%	1.70	72.05	-	43.62	-
		UAP	96.16%	6.33	97.11	36.32%	56.77	30.77%
		NC	99.29%	34.34	75.51	4.67%	45.21	3.96%
		Iterative NC	98.30%	25.37	113.35	53.38%	72.18	60.74%
		Ours	98.30%	27.03	115.63	55.96%	75.58	67.86%

ened by our method are more resilient to existing natural backdoor attacks.

L.2. Backdoor Scanning

Backdoor scanning aims to scan a given model to decide if it contains a backdoor, without assuming any inputs stamped with the backdoor pattern [19, 21, 22, 26, 52, 63, 68, 72, 76]. This is one of the popular defense solutions against backdoor attack. Many existing backdoor scanners are built on top of NC's trigger generation method. For instance, a state-of-the-art approach K-arm [59] uses NC as the base optimization method to generation backdoor. It iteratively and stochastically selects the most promising labels (potentially poisoned) for optimization with the guidance of an objective function. It achieves the top performance on the TrojAI competition organized by IARPA [50], outperforming NC [65], ABS [38], TABOR [18], DLTND [68], and other existing methods. To evaluate the performance of our backdoor generation method in downstream applications, we replace the NC method with ours in the K-arm scanner. We conduct the experiment on 300 pre-trained models from the TrojAI competition. Detailed setup can be found in Appendix C. For a fair comparison, we use the same setting in K-arm (including the pre-selection and the scheduler) and only replace the optimization component. The comparison results are shown in Table 8. We also include the results of a few other baselines from the K-arm paper [59]. The first column denotes detection methods. The following six columns present detection accuracy and time (per model in seconds) on different rounds. Observe that using our method can further boost the performance of K-arm for 1% on round 3 and 2% on round 4, surpassing the state-of-the-art results. Note that since the original K-arm already has high accuracy, the room to improve is small. The time cost is comparable using our method. We also compare our method with another detection approach GangSweep [79] on the TrojAI round 3. We randomly select 20 benign models and 20 poisoned models to conduct the experiment. The

Table 8. Scanning backdoored models on the TrojAI dataset

Method	Round 2		Round 3		Round 4	
	Acc.	Time(s)	Acc.	Time(s)	Acc.	Time(s)
ABS	62%	1527	71%	1435	79%	525
TABOR	55%	$>$ 32000	60%	$>$ 30000	60%	$>$ 35000
DLTND	60%	$>$ 26000	65%	$>$ 29000	65%	$>$ 31000
K-arm	85%	210	91%	183	87%	292
Ours	85%	231	92%	198	89%	320

detection accuracy of GangSweep is only 57.50%, much lower than ours (92%).

We further evaluate our method on detecting three advanced backdoor attacks, namely, WaNet [48], invisible backdoor [32], and blind Backdoor [2].

WaNet [48] uses distortion transformation (e.g., distorting straight lines) as the backdoor. At the pixel level, the backdoor varies for different inputs. We conduct an experiment on backdoored models downloaded from the official repository [48], which are trained on MNIST, CIFAR-10, GTSRB and CelebA, respectively. We use NC and our method to reverse engineer triggers for these models. We use the anomaly index to analyze generated triggers for backdoor detection as in the original NC paper [65]. A large index means that the generated trigger for a label is much smaller than those for other labels. A model with an anomaly index larger than 2 is considered backdoored (i.e., the default setting). Table 3 in Section 5.5 shows the anomaly indices for different models using NC and ours. We can see that NC cannot detect any of the evaluated models (consistent with the results reported in [48]), whereas our method can detect all the backdoored models (as we can generate a much smaller trigger for the target). Our inspection shows that although the injected triggers are pervasive, the models pick up low level features such as curly lines during poisoning. NC generates large triggers for the target class that are not distinguishable from those of benign classes, whereas our triggers are much smaller.

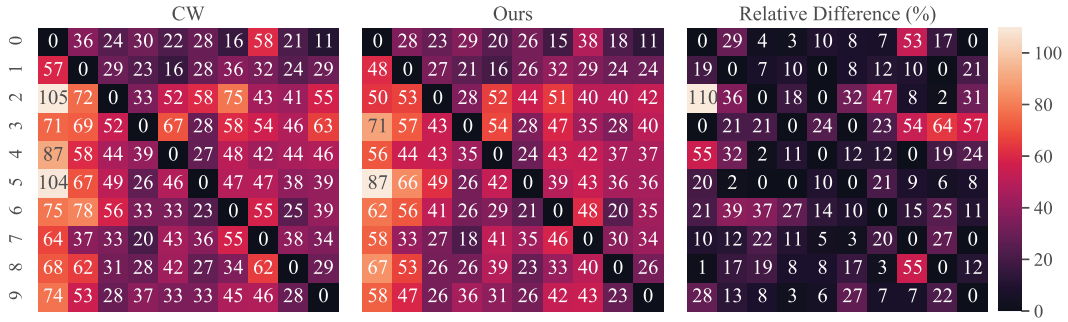
The invisible backdoor [32] uses a uniform perturbation with the smallest L^2 as the backdoor, which is hence pervasive. Since the authors did not provide the implementation or backdoored models in the paper, we have contacted the authors but haven't heard from them yet. We also tried to re-implement their attack based on the paper. With limited details of hyper parameters, the backdoor we got is larger than what was reported in the paper. We hence tested on a backdoored model on CIFAR-10 with the smallest L^2 backdoor that we can get. The normal test accuracy is 90.96% and the ASR is 99.63%. We then use NC and our method to evaluate this model. The anomaly index is 0.77 by NC and 2.28 by ours. The result shows that our method can detect the model as backdoored, whereas NC cannot.

The blind backdoor attack [2] can evade the detection of NC. We study our method against such a strong attack. We use the official repository from the original paper [2] to conduct an experiment. We use the MNIST dataset and replace the optimization of NC with ours in the robust training. After training, we run our method on the trained model to generate triggers for all classes, and use anomaly detection to see whether the model is backdoored. For the above robustly trained model, our method has an anomaly index of 3.37, which can detect the model as backdoored. It indicates that the smoother loss function in our technique allows finding the true trigger despite the robust training. We further demonstrate that when applying our method on the robust model trained with NC (the same in [2]), we can also detect the model as backdoored with an anomaly index of 3.04.

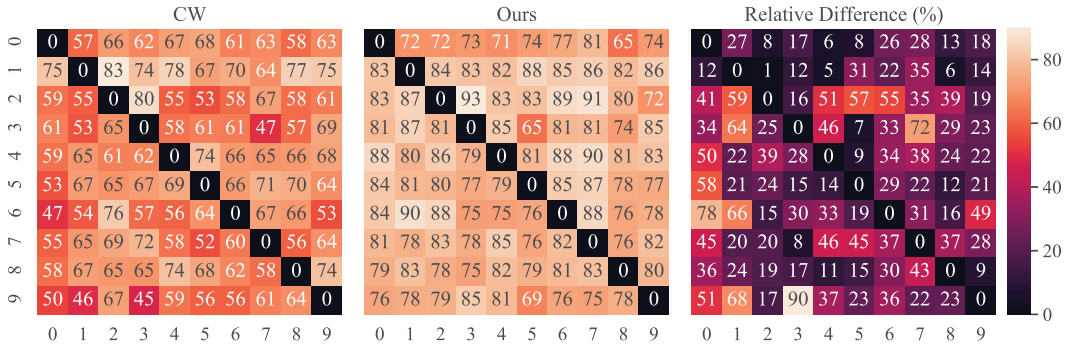
Table 9. Comparison of different methods on more class pairs from ImageNet. The first column shows the victim→target class pairs. The second column shows the methods. The third column is the time cost (in minutes) and the fourth column the number of perturbed pixels (#Pixels). The last column shows ASR on the samples from the validation set.

Pair	Method	Time (min)	#Pixels	ASR
Bullfrog→Robin	UAP	10.28	50175	0.00%
	NC	9.20	26016	28.00%
	Ours	3.69	489	50.00%
Gorilla→Tench	UAP	12.37	50173	0.00%
	NC	9.19	26803	10.00%
	Ours	3.78	587	50.00%
Gorilla→Goldfish	UAP	12.63	50174	0.00%
	NC	9.19	26532	50.00%
	Ours	4.20	626	62.00%
Gorilla→Hammerhead	UAP	12.79	50173	0.00%
	NC	9.11	26854	30.00%
	Ours	4.54	643	54.00%
Treefrog→Goldfish	UAP	13.21	50175	44.00%
	NC	9.20	26714	48.00%
	Ours	4.04	551	58.00%

Pair	Method	Time (min)	#Pixels	ASR
Treefrog→Tiger Shark	UAP	12.70	50175	14.00%
	NC	9.21	27383	28.00%
	Ours	4.12	967	58.00%
Peacock→Ostrich	UAP	13.04	50175	0.00%
	NC	9.47	29800	12.00%
	Ours	4.29	1447	54.00%
Peacock→Bulbul	UAP	12.70	50171	0.00%
	NC	9.37	28882	50.00%
	Ours	5.02	1512	62.00%
Chihuahua→Partridge	UAP	10.45	50175	0.00%
	NC	9.19	26228	36.00%
	Ours	3.77	666	54.00%
Chihuahua→Isopod	UAP	10.35	50173	8.00%
	NC	9.27	26397	26.00%
	Ours	3.69	626	56.00%



(a) #Pixels of generated triggers. The last heat map shows how much larger CW triggers are.



(b) ASRs of generated backdoors. The last heat map shows how much higher our ASRs are.

Figure 12. Comparison of CW and ours for all class pairs on SVHN. Each cell denotes the result of a generated trigger from a victim class (row) to a target class (column). The first two heat maps in each subfigure illustrate the results for CW and ours. The last shows the relative difference.

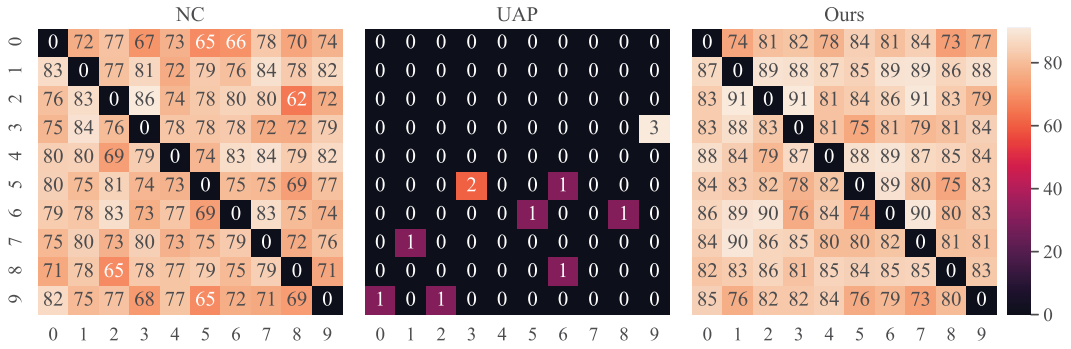
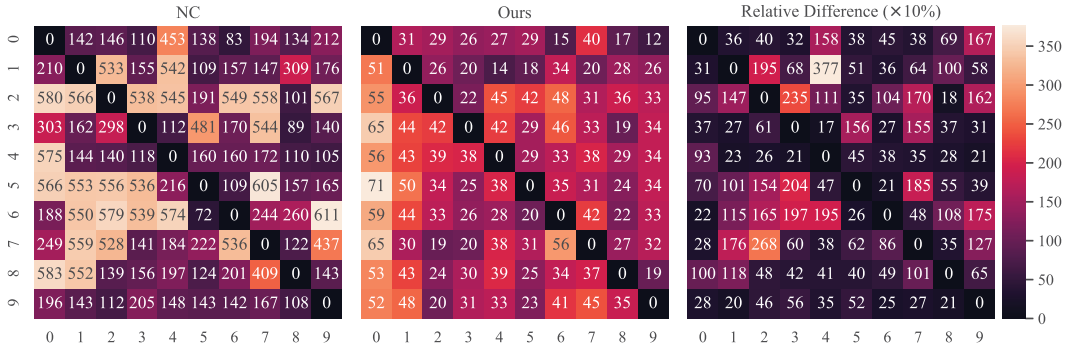
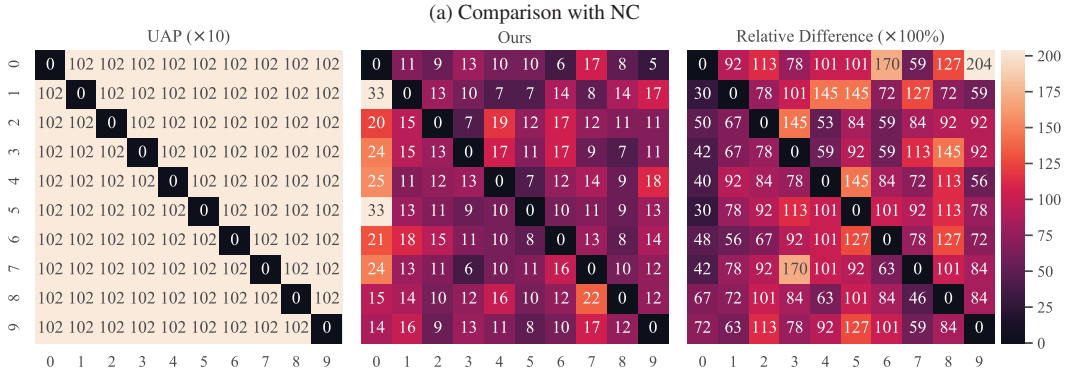


Figure 13. Comparison of NC, UAP and ours on the ASR for all class pairs on the SVHN dataset



(a) Comparison with NC



(b) Comparison with UAP

Figure 14. Comparison of the number of perturbed pixels with the same ASR for all class pairs on the SVHN dataset. The first two heat maps in each subfigure illustrate the results for NC/UAP and ours, respectively. The last heat map shows how much larger of generated backdoors by NC/UAP compared to ours.