Towards Robust Model Watermark via Reducing Parametric Vulnerability

Guanhao Gan[1], Yiming Li[1,4], Dongxian Wu[2,*], Shu-Tao Xia[1,3,*]
[1]Tsinghua Shenzhen International Graduate School, Tsinghua University, China
[2]The University of Tokyo, Japan
[3]Research Center of Artificial Intelligence, Peng Cheng Laboratory, China
[4]Ant Group, China
{ggh21,li-ym18}@mails.tinghua.edu.cn;
d.wu@k.u-tokyo.ac.jp; xiast@sz.tsinghua.edu.cn

Abstract

Deep neural networks are valuable assets considering their commercial benefits and huge demands for costly annotation and computation resources. To protect the copyright of DNNs, backdoor-based ownership verification becomes popular recently, in which the model owner can watermark the model by embedding a specific backdoor behavior before releasing it. The defenders (usually the model owners) can identify whether a suspicious third-party model is "stolen" from them based on the presence of the behavior. Unfortunately, these watermarks are proven to be vulnerable to removal attacks even like fine-tuning. To further explore this vulnerability, we investigate the parameter space and find there exist many watermark-removed models in the vicinity of the watermarked one, which may be easily used by removal attacks. Inspired by this finding, we propose a mini-max formulation to find these watermark-removed models and recover their watermark behavior. Extensive experiments demonstrate that our method improves the robustness of the model watermarking against parametric changes and numerous watermark-removal attacks. The codes for reproducing our main experiments are available at `https://github.com/GuanhaoGan/robust-model-watermarking`.

1. Introduction

While deep neural networks (DNNs) achieve great success in many applications [20, 9, 39] and bring substantial commercial benefits [31, 12, 18], training such a deep model usually requires a huge amount of well-annotated data, massive computational resources, and careful tuning of hyper-parameters. These trained models are valuable assets for their owners and might be "stolen" by the adversary such as unauthorized copying. In many practical scenarios, such as limited open-sourcing [55] (*e.g.*, only for non-commercial purposes) and model trading[1], the model's parameters are directly exposed, and the adversary can simply steal the model by copying its parameters. How to properly protect these trained DNNs is significant.

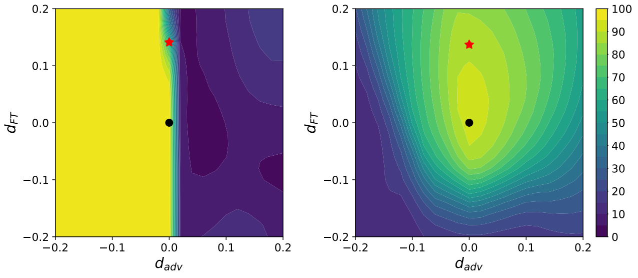
To protect the intellectual property (IP) embodied inside DNNs, several watermarking methods were proposed [45, 10, 35, 5, 29, 49]. Among them, backdoor-based ownership verification is one of the most popular methods [1, 54, 22, 30]. Before releasing the protected DNN, the defender embeds some distinctive behaviors, such as predicting a pre-defined label for any images with "TEST" (watermark samples) as shown in Figure 4. Based on the presence of these distinctive behaviors, the defender can determine whether a suspicious third-party DNN was "stolen" from the protected DNN. The more likely a DNN predicts watermark samples as the pre-defined target label (*i.e.*, with a higher watermark success rate), the more suspicious it is of being an unauthorized copy of the protected model.

However, the backdoor-based watermarking is vulnerable to simple removal attacks [34, 41, 16]. For example, watermark behaviors can be easily erased by fine-tuning[2] with a medium learning rate like 0.01 (see Figure A17 in Zhao *et al*. [56]). To explore such a vulnerability, considering that fine-tuning regards the watermarked model as the start point and continues to update its parameters on some clean data, we investigate how the watermark success rate (WSR) / benign accuracy (BA) changes in the vicinity of the watermarked model in the parameter space. For easier comparison, we use the relative distance $\|\boldsymbol{\theta} - \boldsymbol{\theta}_w\|_2 / \|\boldsymbol{\theta}_w\|_2$ in

[*] Correspondence to: Dongxian Wu (d.wu@k.u-tokyo.ac.jp) and Shu-Tao Xia (xiast@sz.tsinghua.edu.cn).

[1] People are allowed to buy and sell pre-trained models on platforms like AWS marketplace or BigML.

[2] While many watermark methods were believed to be resistant to fine-tuning, they were only tested with small learning rates. For example, Bansal *et al*. [3] only used a learning rate of 0.001 or even 0.0001.



(a) Watermark Success Rate (b) Benign Accuracy

Figure 1. The performance of models in the vicinity of the watermarked model in the parameter space. d_{FT} is the direction of fine-tuning and d_{adv} is the adversarial direction. *black dot*: the original watermarked model; *red star*: the model after fine-tuning.

the parameter space, where $\boldsymbol{\theta}_w$ is the original watermarked model and corresponds to the origin in the coordinate axes (the black circle), for discussions. As shown in Figure 1, we find that fine-tuning on clean data (black circle → red star) changes the model with 0.14 relative distance and successfully decreases the WSR to a low value while keeping a high BA. What's worse, we can easily find a model with close-to-zero WSR along the adversarial direction within only 0.03 relative distance[3]. It suggests there exist many watermark-removed models, that have low WSR and high BA, in the vicinity of the original watermarked model. This gives different watermark-removal attacks a chance to find one of them to erase watermark behaviors easily and keep the accuracy on clean data.

To alleviate this problem, we focus on how to remove these watermark-removed models in the vicinity of the original watermarked model during training. Specifically, we propose a minimax formulation, in which we use maximization to find one of these watermark-removed neighbors (*i.e.*, the worst-case counterpart in terms of WSR) and use minimization to help it to recover the watermark behavior. Further, when combing our method with prevailing BatchNorm-based DNNs, we propose to use clean data to normalize the watermark samples within BatchNorm during training to mitigate the domain shift between defenses and attacks. Our main contributions are three-fold:

- We demonstrate that there exist many watermark-removed models in the vicinity of the watermarked model in the parameter space, which may be easily utilized by fine-tuning and other removal methods.

- We propose a minimax formulation to find watermark-removed models in the vicinity and recover their watermark behaviors, to mitigate the vulnerability in the parameter space. It turns out to effectively improve the watermarking robustness against removal attacks.

- We conduct extensive experiments against several state-of-the-art watermark-remove attacks to demonstrate the effectiveness of our method. In addition, we also conduct some exploratory experiments to have a closer look at our method.

2. Related Works

Model Watermark and Verification. Model watermark is a common method to design ownership verification for protecting the intellectual property (IP) embodied inside DNNs. The defender first watermarks the model by embedding some distinctive behaviors into the protected model during training. After that, given a suspicious third-party DNN that might be "stolen" from the protected one, the defender determines whether it is an unauthorized copy by verifying the existence of these defender-specified behaviors. In general, existing watermark methods can be categorized into two main types, including *white-box watermark* and *black-box watermark*, based on whether defenders can access the source files of suspicious models. Currently, most of the existing white-box methods [4, 44, 45] embedded the watermark into specific weights or the model activation [7]. These methods have promising performance since defenders can exploit useful information contained in model source files. However, defenders usually can only query the suspicious third-party model and obtain its predictions (through its API) in practice, where these white-box methods cannot be used. In contrast, black-box methods only require model predictions. Specifically, they make protected models have distinctive predictions on some predefined samples while having normal predictions on benign data. For example, Zhang *et al.* [54] and Adi *et al.* [1] watermarked DNNs with backdoor samples [23, 27], while Le *et al.* [21] and Lukas *et al.* [35] exploited adversarial samples [43]. In this paper, we focus on backdoor-based watermark, as it is one of the mainstream black-box methods.

Watermark-removal Attack and Defense. While model owners use many watermark-based techniques to protect their models, adversaries are aware of these methods and attempt to remove them before deploying models. For example, the adversaries can remove the trigger pattern before feeding images into the DNNs [32, 52, 28], or extract the model functionality without inheriting the watermarks via distillation [14, 41]. Amongst them, model modification is the most promising method, achieving satisfactory performance and acceptable computation budgets. Specifically, some methods eliminated watermark-related neurons like fine-pruning (FP) [33] and adversarial neuron perturbation (ANP) [50], while others adapted the model weights according to separate clean data like neural attention distillation (NAD) [25], fine-tuning (FT) [45], and mode connectivity repair (MCR) [56]. As a result, the

[3] Details about the visualization method can be found in Appendix A.

model owners must enhance the robustness of their watermarks against these powerful watermark-removal attacks in black-box verification scenarios. Recently, to make the watermark less sensitive to parameter changes, Namba *et al.* [37] proposed exponentially weighting (EW) model parameters when embedding the watermark. Inspired by the randomized smoothing [6], Bansal *et al.* [3] proposed the certified watermark (CW) by adding Gaussian noise to the model parameters during training and conducting verification in white-box cases, which requires access to model parameters. Instead, we only apply the same training scheme and conduct black-box verification for a fair comparison, which is also applied in Bansal *et al.* [3].

3. The Proposed Method

3.1. Preliminaries

Threat Model. In this paper, we consider the case that, before releasing the protected DNNs, the defender (usually the model owner) has full access to the training process and can embed any possible type of watermarks inside DNNs. For verification, the defender is only able to obtain predictions from the suspicious third-party model via its API (*i.e.*, black-box verification setting). This setting is more practical but also more challenging than the white-box setting where defenders can access model weights.

Deep Neural Network. In this paper, we consider a classification problem with K classes. The DNN model $f_{\boldsymbol{\theta}}$ with its parameters $\boldsymbol{\theta}$ are learned on a clean training dataset $\mathcal{D}_c = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N)\}$, which contains N inputs $x_i \in \mathbb{R}^d, i = 1, \cdots, N$, and the corresponding ground-truth label $y_i \in \{1, \cdots, K\}$. The training procedure tries to find the optimal model parameters to minimize the training loss on the training data $\mathcal{D}_c$, *i.e.*,

$$\mathcal{L}(\boldsymbol{\theta}, \mathcal{D}_c) = \mathop{\mathbb{E}}_{\boldsymbol{x},y\sim\mathcal{D}_c} \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}), y), \tag{1}$$

where $\ell(\cdot, \cdot)$ is usually cross-entropy loss.

Embedding Model Watermark. Defenders are able to inject watermark behaviors during training by using a watermarked dataset $\mathcal{D}_w = \{(\boldsymbol{x}'_1, y'_1), \cdots, (\boldsymbol{x}'_M, y'_M)\}$ containing M pairs of watermark samples and their corresponding label. For example, if expecting the model to always predict class "0" for any input with "TEST", we add "TEST" on a clean image $\boldsymbol{x}_i$ to obtain the watermark sample $\boldsymbol{x}'_i$, and label it as class "0" ($y'_i = 0$). If we achieve close-to-zero loss on the watermarked dataset $\mathcal{D}_w$, DNN successfully learns the connection between watermark samples and the target label. Thus, the training procedure with watermark embedding attempts to find the optimal model parameters to minimize the training loss on both clean training dataset $\mathcal{D}_c$ and watermarked dataset $\mathcal{D}_w$, as follows:

$$\begin{aligned}&\mathcal{L}(\boldsymbol{\theta}, \mathcal{D}_c) + \alpha \cdot \mathcal{L}(\boldsymbol{\theta}, \mathcal{D}_w)\\ =&\mathop{\mathbb{E}}_{\boldsymbol{x},y\sim\mathcal{D}_c} \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}), y) + \alpha \cdot \mathop{\mathbb{E}}_{\boldsymbol{x}',y'\sim\mathcal{D}_w} \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}'), y').\end{aligned} \tag{2}$$

3.2. Adversarial Parametric Perturbation (APP)

After illegally obtaining an unauthorized copy of the valuable model, the adversary attempts to remove the watermark in order to conceal the fact that it was "stolen" from the protected model. For example, the adversary starts from the original watermarked model $f_{\boldsymbol{\theta}_w}(\cdot)$ and continues to update its parameters using clean data. If there exist many models $f_{\boldsymbol{\theta}}(\cdot), \boldsymbol{\theta} \neq \boldsymbol{\theta}_w$, with a low WSR and high BA in the vicinity of the watermarked model as shown in Figure 1, the adversary could easily find one of them and escape the watermark detection from the defender.

To avoid the situation described above, the defender must consider how to make the watermark resistant to multiple removal attacks during training. Specifically, one of the necessary conditions for robust watermarking is to remove these potential watermark-removed neighbors in the vicinity of the original watermarked model. Thus, a robust watermark embedding scheme can be divided into two steps: **(1)** finding watermark-removed neighbors and **(2)** recovering their watermark behaviors.

Maximization to Find the Watermark-removed Counterparts. Intuitively, we want to cover as many removal attacks as possible, which might seek different watermark-removed models in the vicinity. Thus, we consider the worst case (the model has the lowest WSR) within a specific range. Given a feasible perturbation region $\mathcal{B} \triangleq \{\boldsymbol{\delta} \mid \|\boldsymbol{\delta}\|_2 \leq \epsilon \|\boldsymbol{\theta}\|_2\}$, where $\epsilon > 0$ is a given perturbation budget, we attempt to find an adversarial parametric perturbation $\boldsymbol{\delta}$,

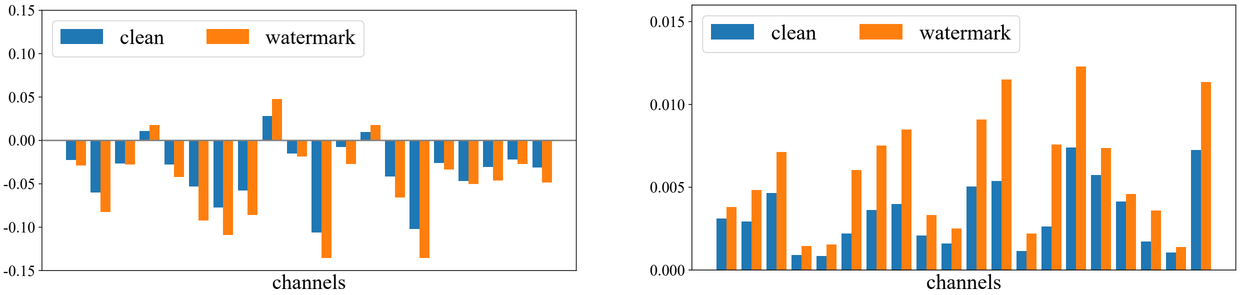
$$\boldsymbol{\delta} \leftarrow \max_{\boldsymbol{\delta}\in\mathcal{B}} \mathcal{L}(\boldsymbol{\theta} + \boldsymbol{\delta}, \mathcal{D}_w). \tag{3}$$

In general, $\boldsymbol{\delta}$ is the worst-case weight perturbation that can be added to the watermarked model for generating its perturbed version $f_{\boldsymbol{\theta}+\boldsymbol{\delta}}(\cdot)$ with low watermark success rate.

Minimization to Recover the Watermark Behaviors. After seeking the worst case in the vicinity, we should reduce the training loss on watermark samples of the perturbed model $f_{\boldsymbol{\theta}+\boldsymbol{\delta}}(\cdot)$ to recover its watermark behavior. Meanwhile, we always expect the model $f_{\boldsymbol{\theta}}(\cdot)$ to have low training loss on the clean training data to have satisfactory utility. Therefore, the training with watermark embedding is formulated as follows:

$$\min_{\boldsymbol{\theta}} \big[\mathcal{L}(\boldsymbol{\theta}, \mathcal{D}_c) + \alpha \cdot \max_{\boldsymbol{\delta}\in\mathcal{B}} \mathcal{L}(\boldsymbol{\theta} + \boldsymbol{\delta}, \mathcal{D}_w)\big]. \tag{4}$$

The Perturbation Generation. However, since DNN is severely non-convex, it is impossible to solve the maximization problem accurately. Here, we apply a single-step



(a) The estimation of running mean

(b) The estimation of running variance

Figure 2. The distribution for clean samples and watermark samples on CIFAR-10.

method to approximate the worst-case perturbation. Besides, the perturbation magnitude varies across architectures. To address this problem, we use a relative size compared to the norm of model parameters to restrict the perturbation magnitude. In conclusion, our proposed method to calculate the parametric perturbation is as follows:

$$\boldsymbol{\delta} \leftarrow \epsilon \|\boldsymbol{\theta}\|_2 \cdot \frac{\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \mathcal{D}_w)}{\|\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \mathcal{D}_w)\|_2}, \tag{5}$$

where $\frac{\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \mathcal{D}_w)}{\|\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \mathcal{D}_w)\|_2}$ is the normalized direction vector whose length equals 1, and $\epsilon \|\boldsymbol{\theta}\|_2$ controls the magnitude of the perturbation in a relative way.

3.3. Estimate BatchNorm Statistics on Clean Inputs

The Assumption of Domain Shift. In preliminary experiments, we find our proposed algorithm cannot improve the robustness of the watermark (see Table 3). We conjecture this failure is caused by the domain shift between the defense and attacks. Specifically, we only feed watermark samples into DNN, and all inputs of each layer are normalized by statistics from them when computing the adversarial perturbation and recovering the watermark behavior. In other words, the defender conducts the watermark embedding in the domain of watermark samples. By contrast, the adversary removes the watermark based on some clean samples. A similar problem about domain shift is also observed in domain adaption [26].

The Verification of Domain Shift. To verify the aforementioned assumption, we analyze the estimated mean and variance inside BatchNorm for clean samples and watermark samples. We visualize these estimations of different channels in the 9-th layer of ResNet-18 [13] on CIFAR-10 [19], and set the images with "TEST" as the watermark samples for the discussion. As shown in Figure 2, there is a significant discrepancy between clean samples (the blue bar) and watermark samples (the orange bar). Since vanilla APP is performed using watermark samples while the attacker removes the watermark using clean samples, the discrepancy between clean and watermark samples may hinder the robustness of the watermark behavior.

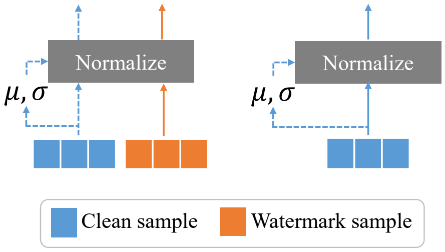


Figure 3. The diagram of our c-BN. We use BatchNorm statistics from the clean inputs to normalize the watermark inputs.

The Proposed Customized BatchNorm. To reduce the discrepancy, we propose **c**lean-sample-based **B**atch**N**orm (c-BN). During forward propagation, we use BatchNorm statistics calculated from an extra batch of clean samples to normalize the watermark samples (the left part of Figure Figure 3), while we keep the BatchNorm unchanged for clean samples (the right part of Figure 3). In the implementation, since we always have a batch of clean samples $\mathcal{B}_c$ and a batch of watermark samples $\mathcal{B}_w$ for each update of model parameters, we always calculate the BatchNorm statistics and normalize inputs for each layer based on the clean batch $\mathcal{B}_c$. Thus, our APP-based watermarking training with c-BN can be reformulated as follows:

$$\min_{\boldsymbol{\theta}} \left[\mathcal{L}(\boldsymbol{\theta}, \mathcal{D}_c) + \alpha \cdot \max_{\boldsymbol{\delta} \in \mathcal{B}} \mathcal{L}(\boldsymbol{\theta} + \boldsymbol{\delta}, \mathcal{D}_w; \mathcal{D}_c)) \right], \tag{6}$$

where $\mathcal{L}(\cdot, \cdot; \mathcal{D}_c)$ denotes that, when calculating this loss term, we use clean samples to estimate batch statistics during forward propagation in c-BN.

3.4. The Overall Algorithm

Here, we introduce the final algorithm of our method, which consists of adversarial parametric perturbation (APP) and clean-sample-based BatchNorm (c-BN). The pseudo-

Algorithm 1 Training APP-based watermarked model.

```
Input: Network $f_{\boldsymbol{\theta}}(\cdot)$, clean training set $\mathcal{D}_c$, watermarked training set $\mathcal{D}_w$, batch size $n$ for clean data, batch size $m$ watermarked data, learning rate $\eta$, perturbation magnitude $\epsilon$
1: Initialize model parameters $\boldsymbol{\theta}$
2: repeat
3:     Sample mini-batch $\mathcal{B}_c = \{(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_n, y_n)\}$ from $\mathcal{D}_c$
4:     $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \mathcal{B}_c)$
5:     Sample mini-batch $\mathcal{B}_w = \{(\boldsymbol{x}'_1, y'_1), \cdots, (\boldsymbol{x}'_m, y'_n)\}$ from $\mathcal{D}_w$
6:     $\boldsymbol{\delta} \leftarrow \epsilon \|\boldsymbol{\theta}\|_2 \frac{\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \mathcal{B}_w; \mathcal{B}_c)}{\|\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \mathcal{B}_w; \mathcal{B}_c)\|}$
7:     $\boldsymbol{g} \leftarrow \boldsymbol{g} + \nabla_{\boldsymbol{\theta}} [\alpha \mathcal{L}(\boldsymbol{\theta} + \boldsymbol{\delta}, \mathcal{B}_w; \mathcal{B}_c)]$   // $\mathcal{L}(\cdot, \cdot; \mathcal{D}_c)$ denotes that, clean samples are used to estimate batch statistics during forward propagation in c-BN.
8:     $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \boldsymbol{g}$
9: until training converged
Output: Watermarked network $f_{\boldsymbol{\theta}}(\cdot)$
```

code of our method can be found in Algorithm 1. Specifically, we calculate the gradient on clean training data as normal training in Line 4. In Line 6, we calculate the APP using clean batch statistics estimated by c-BN. Based on the APP, we calculate the gradient of the perturbed model on the watermarked data and add it to the gradient from clean data in Line 7. We update the model parameters in Line 8, and repeat the above steps until training converges.

4. Experiments

In this section, we conduct comprehensive experiments to evaluate the effectiveness of our proposed method, including a comparison with other watermark embedding schemes, ablation studies, and some exploratory experiments to understand our proposed method.

4.1. Experiment Settings

Dataset Preparation. We conduct experiments on CIFAR-10 and CIFAR-100 [19]. To verify the effectiveness on more practical scenarios, we also do experiments on a subset of the ImageNet [8] dataset, containing 100 classes with 50,000 images for training (500 images per class) and 5,000 images for testing (50 images per class). Similar to Zhang *et al.* [54], we consider three types of watermark samples: 1) Content: adding extra meaningful content to normal images ("TEST" in our experiments). 2) Noise: adding a meaningless randomly-generated noise into normal images; 3) Unrelated: using images from an unrelated domain (SVHN [38] in our experiments). Figure 4 visualizes samples for different watermark types. We set '0' as the target label, *i.e.*, the watermarked DNN always predicts watermark samples as class "airplane" on CFIAR-10 and as "beaver" on CIFAR-100. We use 80% of the original training data to train the watermarked DNNs and use the remaining 20% for potential watermark-removal attacks. Before training, we replace 1% of the current training data as the watermark samples.

Settings for Watermarked DNNs. We train a ResNet-18 [13] for 100 epochs with an initial learning rate of 0.1 and weight decay of 5×10^{-4}. The learning rate is multiplied by 0.1 at the 50-th and 75-th epoch. To train watermarked DNNs, we use our method and several state-of-the-art baselines: 1) *vanilla* watermarking training [54]; 2) exponentialized weight (EW) method [37]; 3) the empirical verification[4] from certified watermarking (CW) [3]. For our APP, we set the coefficient for watermark loss $\alpha = 0.01$ and the maximum perturbation size $\epsilon = 0.02$ on CIFAR-10 and CIFAR-100, and $\epsilon = 0.01$ on ImageNet. Unless otherwise specified, we always use our c-BN during training.

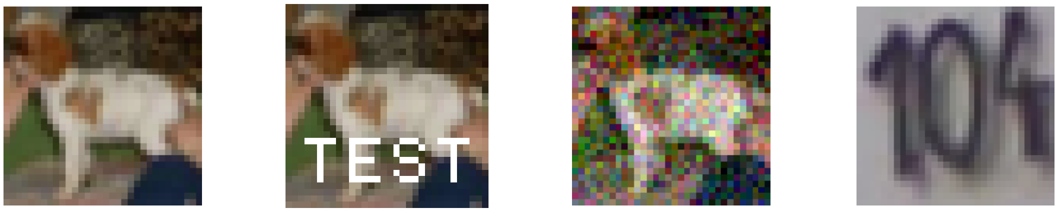
Settings for Removal Attacks. We evaluate the robustness of the watermarked DNN against several state-of-the-art watermark-removal attacks, including: 1) fine-tuning (FT) [45]; 2) fine-pruning (FP) [33]; 3) adversarial neural pruning (ANP) [50]; 4) neural attention distillation (NAD) [25]; 5) mode connectivity repair (MCR) [56]; 6) neural network laundering (NNL) [2]. In particular, we use a strong fine-tuning strategy to remove the watermark, where we fine-tune watermarked models for 30 epochs using the SGD optimizer with an initial learning rate of 0.05 and a momentum of 0.9. The learning rate is multiplied by 0.5 every 5 epochs. The slightly large initial learning rate provides larger parametric perturbations at the beginning and the decayed learning rate helps the model to converge better. More details about FT and other removal methods can be found in Appendix B.4.

Evaluation Metrics. We report the performance mainly on two metrics: **1)** watermark success rate (WSR) on watermark samples, that is the ratio of watermark samples that are classified as the target label by the watermarked DNN and **2)** benign accuracy (BA) on clean test data. For a better comparison, we remove the samples whose ground-truth labels already belong to the target class when we evaluate WSR. In general, an ideal watermark embedding method produces a model with high WSR and high BA, and keeps the high WSR after watermark-removal attacks.

4.2. Main Results

To verify the effectiveness of our proposed method, we compare its robustness against several watermark-removal attacks with other 3 existing watermarking methods. All experiments are repeated over 3 runs with different random

[4] There is also a certified verification in [3], which requires full access to the parameters of the suspicious model. It is out of our scope and we only consider its empirical verification via API.



(a) Original (b) Content (c) Noise (d) Unrelated

Figure 4. The example of different watermark samples.

Table 1. Performance (average over 3 random runs) of 3 watermark-injection methods and 3 types of watermark inputs against 6 removal attacks on CIFAR-10. *Before*: BA/WSR of the trained watermarked models; *After*: the remaining WSR after watermark-removal attacks. *AvgDrop* indicates the average changes in WSR against all attacks.

Type	Method	Before		After						AvgDrop
		BA	WSR	FT	FP	ANP	NAD	MCR	NNL	
Content	*Vanilla*	**93.86**	99.56	56.78	74.58	25.34	48.14	16.56	21.02	↓ 59.15
	EW	92.86	99.17	55.11	63.22	66.24	48.92	25.17	29.15	↓ 51.20
	CW	93.73	99.62	26.98	54.22	27.39	29.18	29.97	19.78	↓ 68.36
	Ours	93.42	**99.87**	**96.63**	**98.44**	**99.56**	**90.76**	**84.65**	**68.58**	↓ **10.10**
Noise	*Vanilla*	93.57	99.99	28.38	28.21	14.52	3.88	10.99	1.00	↓ 85.50
	EW	92.99	99.99	5.10	39.35	28.54	0.04	0.07	**3.34**	↓ 87.25
	CW	**93.67**	**100.00**	0.13	10.87	0.18	0.04	1.41	0.30	↓ 97.84
	Ours	93.47	**100.00**	**66.54**	**75.59**	**83.73**	**23.98**	**68.86**	3.22	↓ **46.35**
Unrelated	*Vanilla*	**93.52**	**100.00**	18.82	24.61	22.31	2.76	10.91	67.35	↓ 75.54
	EW	93.02	99.97	71.46	66.59	46.48	12.48	32.44	64.94	↓ 50.90
	CW	93.47	**100.00**	9.51	14.17	3.20	5.28	5.02	13.41	↓ 91.57
	Ours	93.30	99.95	**96.15**	**95.46**	**99.60**	**89.28**	**87.49**	**94.49**	↓ **6.20**

seeds. Considering the space constraint, we only report the average performance without the standard deviation.

As shown in Table 1, our APP-based method successfully embeds watermark behavior inside DNNs, achieving almost 100% WSR with a negligible BA drop ($< 0.50\%$). Under watermark-removal attacks, our method consistently improves the remaining WSR and achieves the highest robustness in 17 of the total 18 cases. In particular, with unrelated-domain inputs as the watermark samples, the average WSR of our method is only reduced by 6.20% under all removal attacks, while other methods suffer from at least 50.90% drop in WSR. We find that, although NNL is the strongest removal attack (all WSRs decrease below 22%) when watermark samples are those images superimposed by some content or noise, it has an unsatisfactory performance to unrelated-domain inputs as watermark samples[5]. Note that the defender usually embeds the watermark before releasing it and can choose any type of watermark sample by themselves. Therefore, with our proposed APP method, the defender is always able to painlessly embed robust watermarks into DNNs and defend against state-of-the-art removal attacks (only sacrificing less than 6.2% of WSR after attacks). We have similar findings on ImageNet (see Table 2) and CIFAR-100 (see Appendix B.6).

4.3. Ablation Studies

In this section, we conduct several experiments to explore the effect of each part in our proposed methods, including different components, varying perturbation magnitudes, and various target classes. In the following experiments, we always use the images with the content "TEST" as the watermark sample unless otherwise specified.

Effect of Different Components. Our method consists of two parts, *i.e.*, the adversarial parametric perturbation (APP) and the clean-sample-based BatchNorm (c-BN). we evaluate the contribution of each component. We train and evaluate watermarked DNNs without any components (the *Vanilla* method), with one of the components, and with both components (our proposed method). In Table 3, only with APP, we fail in keeping the average WSR under removal attacks due to the domain shift as mentioned in Sec 3.3. Fortunately, with c-BatchNorm, APP solves the domain shift problem and successfully improves the robustness against removal attacks, *e.g.*, it keeps WSR $> 90\%$ against several

[5] This is because NNL first reconstructs the watermark trigger (*e.g.*, the content "TEST" on watermark samples) and then removes watermark behaviors. By contrast, when we use unrelated-domain inputs as watermark samples, there is no trigger pattern, leading to the failure of NNL.

Table 2. Performance (average over 3 random runs) of 3 watermark-injection methods and 3 types of watermark inputs against 6 removal attacks on ImageNet-subset. *Before*: BA/WSR of the trained watermarked models; *After*: the remaining WSR after watermark-removal attacks. *AvgDrop* indicates the average changes in WSR against all attacks.

Type	Method	Before		After						AvgDrop
		BA	WSR	FT	FP	ANP	NAD	MCR	NNL	
Content	*Vanilla*	74.81	98.26	22.18	9.31	43.91	4.40	12.48	28.05	↓ 78.20
	EW	**75.15**	95.85	8.95	3.82	17.07	3.02	8.82	19.96	↓ 85.58
	CW	74.52	99.05	6.35	0.16	0.26	0.68	2.92	17.91	↓ 94.34
	Ours	72.29	**99.54**	**57.56**	**21.46**	**98.57**	**31.95**	**71.93**	**79.39**	↓ **39.40**
Noise	*Vanilla*	74.47	98.65	9.54	2.79	29.00	9.75	8.06	**3.60**	↓ 88.20
	EW	**75.09**	95.36	3.58	4.08	1.19	1.62	4.19	1.56	↓ 92.66
	CW	74.11	98.32	15.35	2.57	11.65	5.65	3.41	2.56	↓ 91.45
	Ours	71.48	**99.38**	**33.80**	**11.69**	**95.52**	**32.54**	**28.40**	1.43	↓ **65.48**
Unrelated	*Vanilla*	74.69	99.97	47.40	36.53	99.66	24.16	54.43	30.87	↓ 51.13
	EW	**75.25**	99.97	33.64	31.12	94.40	59.91	12.94	56.70	↓ 51.85
	CW	74.97	99.99	38.94	0.86	1.97	43.68	65.74	26.66	↓ 70.34
	Ours	73.55	**100.00**	**93.98**	**81.97**	**99.99**	**88.99**	**93.97**	**96.57**	↓ **7.42**

Table 3. The effect of the two components in our method.

APP	c-BN	Before		After						AvgDrop
		BA	WSR	FT	FP	ANP	NAD	MCR	NNL	
		93.86	99.56	56.78	74.58	25.34	48.14	16.56	21.02	↓ 59.15
	✓	**93.94**	99.75	58.14	74.92	10.26	35.17	19.14	23.37	↓ 62.91
✓		93.31	99.69	24.20	38.16	0.91	14.16	19.23	8.03	↓ 82.24
✓	✓	93.42	**99.87**	**96.63**	**98.44**	**99.56**	**90.76**	**84.65**	**68.58**	↓ **10.10**

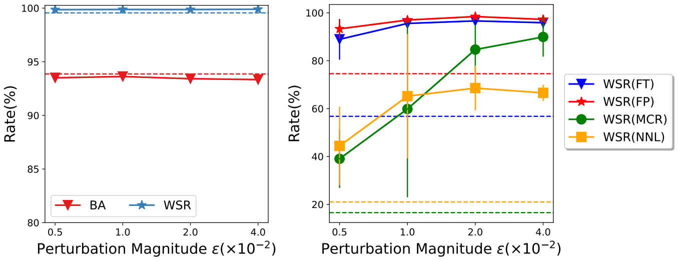


Figure 5. The results with various magnitude ϵ. We use the dashed line with the same color to show the performance when $\epsilon = 0$. *Left*: before attacks; *Right*: after attacks.

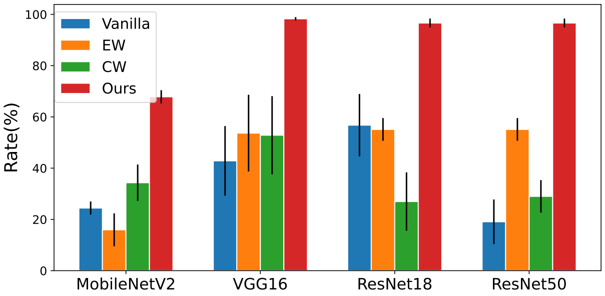
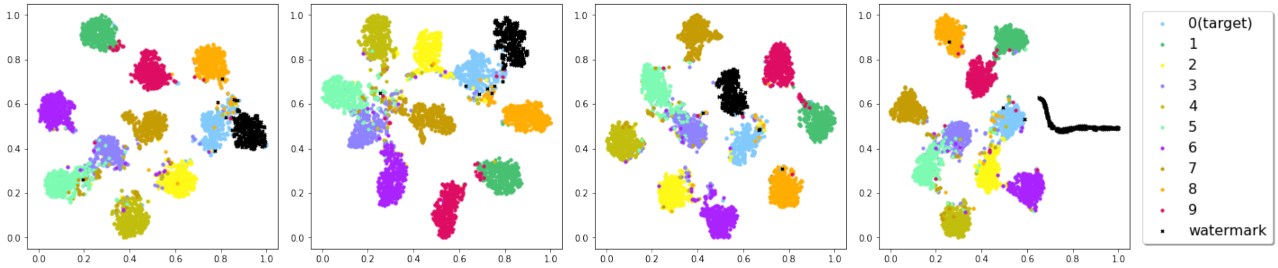


Figure 6. The results of our methods and other baselines with various architectures against FT attack. Our method consistently improves watermark robustness.

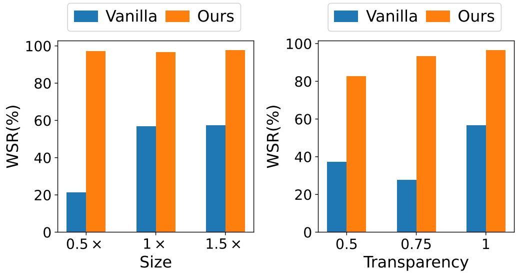
removal attacks (FT, FP, ANP, and NAD), and even keeps WSR 68.58% against the strongest attack NNL. Besides, we find the watermark with only c-BN fails to improve the WSR, which indicates the c-BN just helps APP rather than improving watermark robustness directly. In conclusion, both are essential components contributing to final robustness against watermark-removal attacks.

Effect of Varying Perturbation Magnitude. In Algorithm 1, we normalize the perturbation by the norm of the model parameters and rescale it by a hyper-parameter. Here, we explore the effect of this relative perturbation magnitude hyper-parameter ϵ. We illustrate the performance of the watermarked DNNs before and after removal attacks with varying perturbation magnitude in Figure 5, and find that, within a specific region $\epsilon \leq 4.0 \times 10^{-2}$, our method always improves the robustness against attacks while keeping BA high in a large range for hyperparameter. Besides, we find the selection of hyper-parameter ϵ is more related to the watermark embedding method itself rather than removal attacks (we have similar trends against FT, FP, MCR and NNL). This makes the selection of hyper-parameter ϵ quite straightforward and gives us simple guidance for tuning ϵ in practical scenarios: Although knowing nothing about the potential attack (suppose the adversary applies MCR), the



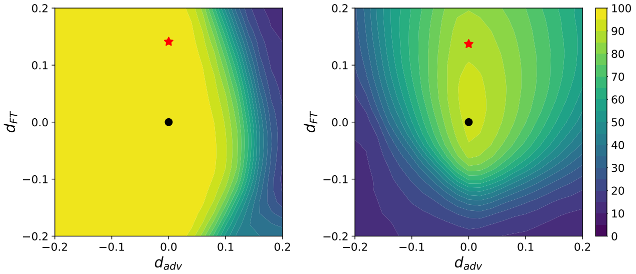
(a) Vanilla (b) EW (c) CW (d) Ours

Figure 7. The t-SNE visualization of hidden feature representations.



(a) Size (b) Transparency

Figure 8. Results with various trigger sizes and transparencies. 1× represents the settings of the original trigger.



(a) Watermark Success Rate (b) Benign Accuracy

Figure 9. The performance of models in the vicinity of APP-based watermarked model in the parameter space. d_{FT} denotes the direction of fine-tuning and d_{adv} denotes the adversarial direction. *black dot*: the original watermarked model; *red star*: the model after fine-tuning.

defender could tune the hyper-parameter against the FT attacks, and the resulting model also achieves satisfactory results against MCR. Detailed results against other attacks can be found in Appendix C.1.

Effect of Various Target Classes. Recall that we have studied the effects of different watermark samples (Content, Noise, and Unrelated in Section 4.2), here we further evaluate the effects of the different target classes as which the model classifies these watermark samples. We set the target class as 1, 2, 3, and 4, respectively. We obtain an average WSR of 94.87%, 79.81%, 84.36% and 87.76% respectively under all removal attacks, while the *vanilla* method only achieves $32.91\%, 20.79\%, 32.28\%,$ and 10.13% (details can be found in Appendix C.2). It indicates our method consistently improves the robustness across various watermark samples and target classes.

Effect on Trigger Size and Transparency. To further verify that our method can apply to triggers with different sizes and transparencies, we also exploit various sizes and transparencies of the "TEST" trigger and evaluate the robustness using FT attack. As shown in Figure 8, our method consistently reaches better performance than the baseline across various trigger sizes and transparencies.

Effect of Different Architectures. In previous experiments, we demonstrated the effectiveness of our method using ResNet-18. Here, we explore the effect of the model architectures across different sizes including MobileNetV2 [40] (a tiny model), VGG16 [42], ResNet-18 and ResNet-50 [13] (a relatively large model) with same hyper-parameters (especially ϵ). As shown in Figure 6, our method always achieves notable improvements ($> 30\%$) compared with other baseline methods in all cases.

4.4. A Closer Look at the APP Method

In this section, we further explore the mechanism of our APP. We visualize the landscape of watermarked model in the parameter space and the distribution of the clean and watermark samples in the feature space for discussions.

The Parameter Space. We start by studying the properties of the watermarked model in the parameter space in the Introduction and illustrate how WSR changes in the vicinity of the watermarked model from the *vanilla* method in Figure 1. Here, we use the same visualization method to show the vicinity of the APP-based method (please see more details in Appendix A). As shown in Figure 9, we find the APP-based watermarked model is able to keep WSR high within a larger range compared to the *vanilla* one. Especially, our model is better in robustness against parametric perturbation along the adversarial direction, which makes it more difficult for the adversary to find watermark-removed models in the vicinity of the protected model.

The Feature Space. To dive into APP, we also visualize the hidden representation of clean samples and watermark samples using the t-SNE method [46] based on different watermark embedding schemes. As shown in Figure 7, in the feature space of our model, the cluster of watermark samples in our method has a larger coverage in the feature space. This may explain why our method is more robust because moving all these watermark samples back to their original clusters takes much more effort. Implementation details and more results can be found in Appendix F.

5. Discussion and Conclusion

In our threat model, we actually limit the parameter perturbation size, *i.e.*, the adversary cannot change the model parameters too much. By contrast, in practice, the adversary is only required to maintain the high benign accuracy of DNNs during watermark-removal attacks. We admit the latter is a better threat model, while it is infeasible to analyze rigorously. It is mostly because we cannot explicitly describe the relationship between benign accuracy and model parameters (we only know some checkpoints and their BA), which prevents its direct usage in the algorithm. Instead, we use a simplified constraint by the perturbation magnitude and believe it is a feasible method: (1) In most cases, attackers use the watermarked model as the initial point and fine-tune model parameters, which (probably) bounds the change of model parameters within a distance; (2) We achieve better robustness against various practical attacks using our threat model. We notice that the defense in our threat model is only a prerequisite for defense in the better threat model. We hope our method can serve as the cornerstone towards truly robust watermarks.

Overall, we investigated the parameter space of watermarked DNNs in this paper. We found that there exist many watermark-removed models in the vicinity of the watermarked model, which may be easily used by removal attacks. To alleviate this problem, we proposed a minimax formulation to find the watermark-removed models and repair their watermark behaviors. In particular, we observed that there is a domain shift between defenses and removal attacks when calculating BatchNorm statistics, based on which we proposed to estimate them only with benign samples (dubbed 'c-BN'). We conducted extensive experiments on benchmark datasets, showing that our method can consistently improves the robustness against several state-of-the-art removal attacks. We hope our method could help model owners better protect their intellectual properties.

Acknowledgement

This work is supported in part by the National Natural Science Foundation of China under Grant 62171248, Shenzhen Science and Technology Program under Grant JCYJ20220818101012025, and the PCNL Key Project under Grant PCL2021A07.

References

[1] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *USENIX Security*, pages 1615–1631, 2018.

[2] William Aiken, Hyoungshick Kim, Simon Woo, and Jungwoo Ryoo. Neural network laundering: Removing black-box backdoor watermarks from deep neural networks. *Computers & Security*, 106:102277, 2021.

[3] Arpit Bansal, Ping-yeh Chiang, Michael J Curry, Rajiv Jain, Curtis Wigington, Varun Manjunatha, John P Dickerson, and Tom Goldstein. Certified neural network watermarks with randomized smoothing. In *ICML*, pages 1450–1465. PMLR, 2022.

[4] Huili Chen, Bita Darvish Rouhani, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. Deepmarks: A secure fingerprinting framework for digital rights management of deep learning models. In *ICMR*, pages 105–113, 2019.

[5] Jialuo Chen, Jingyi Wang, Tinglan Peng, Youcheng Sun, Peng Cheng, Shouling Ji, Xingjun Ma, Bo Li, and Dawn Song. Copy, right? a testing framework for copyright protection of deep learning models. In *IEEE S&P*, pages 824–841. IEEE, 2022.

[6] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In *ICML*, pages 1310–1320. PMLR, 2019.

[7] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. Deepsigns: An end-to-end watermarking framework for ownership protection of deep neural networks. In *ASPLOS*, pages 485–497, 2019.

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255. Ieee, 2009.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[10] Lixin Fan, Kam Woh Ng, and Chee Seng Chan. Rethinking deep neural network ownership verification: embedding passports to defeat ambiguity attacks. In *NeurIPS*, pages 4714–4723, 2019.

[11] Kuofeng Gao, Yang Bai, Jindong Gu, Yong Yang, and Shu-Tao Xia. Backdoor defense via adaptively splitting poisoned dataset. In *CVPR*, pages 4005–4014, 2023.

[12] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(3):362–386, 2020.

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

[14] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[15] Kunzhe Huang, Yiming Li, Baoyuan Wu, Zhan Qin, and Kui Ren. Backdoor defense via decoupling the training process. In *ICLR*, 2021.

[16] Ziheng Huang, Boheng Li, Yan Cai, Run Wang, Shangwei Guo, Liming Fang, Jing Chen, and Lina Wang. What can discriminator do? towards box-free ownership verification of generative adversarial networks. In *ICCV*, 2023.

[17] Kassem Kallas and Teddy Furon. Rose: A robust and secure dnn watermarking. In *2022 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–6. IEEE, 2022.

[18] Veton Kepuska and Gamal Bohouta. Next-generation of virtual personal assistants (microsoft cortana, apple siri, amazon alexa and google home). In *CCWC*, pages 99–103. IEEE, 2018.

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

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, volume 25, 2012.

[21] Erwan Le Merrer, Patrick Perez, and Gilles Trédan. Adversarial frontier stitching for remote neural network watermarking. *Neural Computing and Applications*, 32(13):9233–9244, 2020.

[22] Yiming Li, Yang Bai, Yong Jiang, Yong Yang, Shu-Tao Xia, and Bo Li. Untargeted backdoor watermark: Towards harmless and stealthy dataset copyright protection. In *NeurIPS*, 2022.

[23] Yiming Li, Yong Jiang, Zhifeng Li, and Shu-Tao Xia. Backdoor learning: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[24] Yige Li, Xixiang Lyu, Nodens Koren, Lingjuan Lyu, Bo Li, and Xingjun Ma. Anti-backdoor learning: Training clean models on poisoned data. *NeurIPS*, 34:14900–14912, 2021.

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

[26] Yanghao Li, Naiyan Wang, Jianping Shi, Jiaying Liu, and Xiaodi Hou. Revisiting batch normalization for practical domain adaptation. *arXiv preprint arXiv:1603.04779*, 2016.

[27] Yiming Li, Mengxi Ya, Yang Bai, Yong Jiang, and Shu-Tao Xia. Backdoorbox: A python toolbox for backdoor learning. In *ICLR Workshop*, 2023.

[28] Yiming Li, Tongqing Zhai, Yong Jiang, Zhifeng Li, and Shu-Tao Xia. Backdoor attack in the physical world. In *ICLR Workshop*, 2021.

[29] Yiming Li, Linghui Zhu, Xiaojun Jia, Yong Jiang, Shu-Tao Xia, and Xiaochun Cao. Defending against model stealing via verifying embedded external features. In *AAAI*, 2022.

[30] Yiming Li, Mingyan Zhu, Xue Yang, Yong Jiang, Tao Wei, and Shu-Tao Xia. Black-box dataset ownership verification via backdoor watermarking. *IEEE Transactions on Information Forensics and Security*, 2023.

[31] Zhifeng Li, Dihong Gong, Qiang Li, Dacheng Tao, and Xuelong Li. Mutual component analysis for heterogeneous face recognition. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 7(3):1–23, 2016.

[32] Wei-An Lin, Yogesh Balaji, Pouya Samangouei, and Rama Chellappa. Invert and defend: Model-based approximate inversion of generative adversarial networks for secure inference. *arXiv preprint arXiv:1911.10291*, 2019.

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

[34] Nils Lukas, Edward Jiang, Xinda Li, and Florian Kerschbaum. Sok: How robust is image classification deep neural network watermarking?(extended version). *arXiv preprint arXiv:2108.04974*, 2021.

[35] Nils Lukas, Yuxuan Zhang, and Florian Kerschbaum. Deep neural network fingerprinting by conferrable adversarial examples. In *ICLR*, 2020.

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

[37] Ryota Namba and Jun Sakuma. Robust watermarking of neural network with exponential weighting. In *ACM ASIACCS*, pages 228–240, 2019.

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

[39] Haibo Qiu, Baosheng Yu, Dihong Gong, Zhifeng Li, Wei Liu, and Dacheng Tao. Synface: Face recognition with synthetic data. In *ICCV*, 2021.

[40] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, pages 4510–4520, 2018.

[41] Masoumeh Shafieinejad, Nils Lukas, Jiaqi Wang, Xinda Li, and Florian Kerschbaum. On the robustness of backdoor-based watermarking in deep neural networks. In *IH&MMSec workshop*, pages 177–188, 2021.

[42] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[43] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[44] Enzo Tartaglione, Marco Grangetto, Davide Cavagnino, and Marco Botta. Delving in the loss landscape to embed robust watermarks into neural networks. In *ICPR*, pages 1243–1250. IEEE, 2021.

[45] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. Embedding watermarks into deep neural networks. In *ICMR*, pages 269–277, 2017.

[46] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[47] 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 *IEEE S&P*, pages 707–723. IEEE, 2019.

[48] Lixu Wang, Shichao Xu, Ruiqi Xu, Xiao Wang, and Qi Zhu. Non-transferable learning: A new approach for model own-

ership verification and applicability authorization. In *ICLR*, 2021.

[49] Run Wang, Jixing Ren, Boheng Li, Tianyi She, Wenhui Zhang, Liming Fang, Jing Chen, and Lina Wang. Free fine-tuning: A plug-and-play watermarking scheme for deep neural networks. In *ACM MM*, 2023.

[50] Dongxian Wu and Yisen Wang. Adversarial neuron pruning purifies backdoored deep models. In *NeurIPS*, volume 34, pages 16913–16925, 2021.

[51] Dongxian Wu, Shu-Tao Xia, and Yisen Wang. Adversarial weight perturbation helps robust generalization. In *NeurIPS*, volume 33, pages 2958–2969, 2020.

[52] Valentina Zantedeschi, Maria-Irina Nicolae, and Ambrish Rawat. Efficient defenses against adversarial attacks. In *AISec workshop*, pages 39–49, 2017.

[53] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric Xing, Laurent El Ghaoui, and Michael Jordan. Theoretically principled trade-off between robustness and accuracy. In *ICML*, pages 7472–7482. PMLR, 2019.

[54] Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph Stoecklin, Heqing Huang, and Ian Molloy. Protecting intellectual property of deep neural networks with watermarking. In *ACM ASIACCS*, pages 159–172, 2018.

[55] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

[56] Pu Zhao, Pin-Yu Chen, Payel Das, Karthikeyan Natesan Ramamurthy, and Xue Lin. Bridging mode connectivity in loss landscapes and adversarial robustness. In *ICLR*, 2020.

A. Details about Vicinity Visualization

To visualize the vicinity, we measure the watermark success rate (WSR) and benign accuracy (BA) on the panel spanned by the two directions d_{adv} and d_{FT}. Specifically, d_{adv} is the direction to erase watermark, *i.e.*, $d_{adv} = \nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}, \mathcal{D}_w)$, and d_{FT} is the direction from the original watermarked model $\boldsymbol{\theta}_w$ to a fine-tuned model $\boldsymbol{\theta}_{FT}$, *i.e.*, $d_{FT} = \boldsymbol{\theta}_{FT} - \boldsymbol{\theta}_w$. We fine-tune the original model $\boldsymbol{\theta}_w$ for 40 iterations with the SGD optimizer using a learning rate 0.05 to obtain $\boldsymbol{\theta}_{FT}$. We explore the vicinity by moving the original parameter along with these two directions, recoding WSR and BA of neighbor model. For easier comparison, we use the relative distance in the parametric space, *i.e.*,

$$\boldsymbol{\theta} = \boldsymbol{\theta}_w + \alpha \frac{d_{adv}}{\|d_{adv}\|}\|\boldsymbol{\theta}_w\| + \beta \frac{d_{FT}}{\|d_{FT}\|}\|\boldsymbol{\theta}_w\|, \qquad (7)$$

where (α, β) are the given coordinates. After obtaining the parameter $\boldsymbol{\theta}$ in the vicinity, we further adjust BatchNorm by re-calculating the statistic on the clean dataset to restore benign accuracy. Finally, we evaluate this neighbor model and record its benign accuracy and watermark success rate.

B. Details about Main Experiments

In this section, we first briefly introduce our baseline methods, then provide the detailed settings for our main experiments. We report the full results on CIFAR-10 and CIFAR-100 at the end.

B.1. More details about baseline methods

Vanilla model watermark [54] mixed the watermark samples with the clean samples, based on which to train the model. EW [37] trained the model with exponentially reweighted parameter $EW(\theta, T)$ rather than vanilla weight θ. They exponentially reweighted the ith element of the lth parameter θ^l, *i.e.*,

$$EW(\theta^l, T) = \theta^l_{exp}, \text{ where } \theta^l_{exp,i} = \frac{\exp(|\theta^l_i|T)}{\max_i(\exp(|\theta^l_i|T))}\theta^l_i, \qquad (8)$$

and T is a hyper-parameter adjusting the intensity of the reweighting. As shown in the above equation, the weight elements with a big absolute value will remain almost the same after the reweight operation, while the ones with a small value will decrease to nearly zero. This encourages the neural network to lean on the weights with large absolute values to make decisions, hence making the prediction less sensitive to small weight changes. CW [3] aimed at embedding a watermark with certifiable robustness. They adopted the theory of randomized smoothing [6] and watermarked the network using a gradient estimated with random perturbed weights. The gradient on the watermark batch $\mathcal{B}$ is calculated by

$$g_\theta = \frac{1}{k}\sum_{i=1}^{k} E_{G\in\mathcal{N}(0,(\frac{i}{k})^2 I)} E_{(x,y)\in\mathcal{B}}[\nabla l(x,y;\theta+G)], \qquad (9)$$

where σ is the noise strength.

B.2. Details about Watermark-removal Attacks

Currently, there are some watermark-removal attacks to counter model watermarking. According to Lukas *et al.* [34], existing removal attacks can be divided into three main categories, including 1) *input pre-processing*, 2) *model extraction*, and 3) *model modification*. In general, the first type of attack pre-processes each input sample to remove trigger patterns before feeding it into the deployed model [32]. Model extraction [14, 41] distills the dark knowledge from the victim model to remove distinctive prediction behaviors while preserving its main functionalities. Model modification [45, 33] changes model weights while preserving its main structure. In this paper, we mainly focus on the model-modification-based removal attacks, since input pre-processing has minor benefits for countering backdoor-based watermark [34] and model extraction usually requires a large number of training samples that are inaccessible for defenders in practice [35].

Apart from these traditional watermark attacks mentioned above, we also adopted some backdoor-removal methods to conduct a more thorough evaluation because our watermark method is backdoor-based. The backdoor-removal method can also be derived into two categories, including 1) *post-training backdoor removal methods* [51, 25, 56] that remove backdoor with local benign samples after training, 2) *training-time backdoor removal methods* [24, 15, 11] that directly train a clean model from a poisoned training set. In our experiments, we focus on the *post-training backdoor removal methods* because only the model owner controls the training process.

The description of our adopted watermark/backdoor-removal methods is listed in the following.

FT. Uchida *et al.* [45] removed the watermark by updating model parameters using additional holding clean data.

FP. Liu *et al.* [33] presumed that watermarked neurons are less activated by clean data, and thus pruned the least activated neurons in the last layer before fully-connected layers. They further find-tuned the pruned model to restore benign accuracy and suppress watermarked neurons.

ANP. Wu *et al.* [50] found that backdoored neurons are sensitive to weight perturbation and proposed to prune these neurons to remove the backdoor.

NAD. Li *et al.* [25] utilized knowledge from a fine-tuned model where the watermark is partially removed, to guide the watermark unlearning.

MCR. Zhao *et al.* [56] found that the existence of a high accuracy pathway connecting two backdoored models in the parametric space, and the interpolated model along the path usually doesn't have backdoors. This property allows MCR to be applied in the mission of watermark removal.

NNL. Aiken *et al.* [2] first reconstructed trigger using Neural Cleanse [47], then reset neurons that behave differently on clean data and reconstructed trigger data, and further fine-tuned the model to restore benign accuracy and suppress watermarked neurons.

B.3. More Details about Watermark Settings

Settings for EW. As suggested in its paper [37], we fine-tune a pre-trained model to embed the watermark. We pre-train the model using the original dataset without injecting the watermark samples. The pre-trained model is trained for 100 epochs using the SGD optimizer with an initial learning rate of 0.1, the learning rate decays by a factor of 10 at the 50th and 75th epochs. We fine-tune the pre-trained model for 20 epochs to embed the watermark, with an initial learning rate of 0.1, and the learning rate is drop by 10 at the 10th and 15th epochs.

Settings for CW. For a fair comparison, we adopt a learning rate schedule and a weight-decay factor identical to other methods. Unless otherwise specified, other settings are the same as those used in [3].

Settings for Our Method. For the classification loss term, we calculate the loss using a batch of 128 clean samples, while for the watermark term, we use a batch of 64 clean samples and 64 watermark samples to obtain the estimation of adversarial gradients.

B.4. Details about Watermark-removal Settings

Settings for FT. We fine-tune the watermarked model for 30 epochs using the SGD optimizer with an initial learning rate of 0.05 and a momentum of 0.9, the learning rate is dropped by a factor of 0.5 every five epochs.

Settings for FP. We prune 90% of the least activated neurons in the last layer before fully-connected layers, and after pruning, we fine-tune the pruned model using the same training scheme as FT.

Settings for ANP. We set the pruning rate to 0.6, where all defense shares a similar BA, as shown in Figure 11.

Settings for NAD. The original NAD only experimented on WideResNet models. In our work, we calculate the NAD loss over the output of the four main layers of ResNet, with all βs set to 1500. To obtain a better watermark removal performance, we use an initial learning rate of 0.02 , which is larger than 0.01 in the original paper [25].

Settings for MCR. MCR finds a backdoor-erased model on the path connecting two backdoored models. But in our settings, only one watermarked model is available. Hence the attacker must obtain the other model via fine-tuning the original watermarked model, then perform MCR using the original watermarked model and fine-tuned model. We split the attacker's dataset into two equal halves, one used to fine-tune the model and the other one to train the curve connecting the original model and the fine-tuned model. This fine-tuning is performed for 50 epochs with an initial learning rate of 0.05, which decays by a factor of 0.1 every 10 epochs. For MCR results, $t = 0$ denotes the original model and $t = 1$ denotes the original model. We select results with $t = 0.9$, where all defense shares similar BA, see Figure 10.

Settings for NNL. We reconstruct the trigger using Neural Cleanse [47] for 15 epochs, and reset neurons that behave significantly different under clean input and reconstructed input, we fine-tune the model for 15 epochs with the SGD optimizer, the initial learning rate is 0.02 and is divided by 10 at the 10th epoch.

B.5. Detailed Results on CIFAR-10

The detailed results on CIFAR-10 are shown in Table 4. Moreover, we can observe from Figure 10 and Figure 11 that our method is more robust than other methods, regardless of the threshold value used in MCR and ANP.

B.6. Detailed Results on CIFAR-100

To show that our method can also apply to other datasets, we conduct additional experiments on CIFAR-100.

Modification to Attack Settings. As trigger reconstruction need to scan 100 classes on CIFAR-100, we reduce the NC reconstruction epoch from 15 to 5 to speed it up. The ANP pruning threshold is set to 0.5 in CIFAR-100 experiments to maintain benign accuracy.

Results. As shown in Table 5, similar to previous results on CIFAR-10, our methods generally achieves better watermark robustness compared with other methods. The only exception is on noise watermark where all watermark embedding schemes failed to protect the watermark against FP and NNL attacks. Moreover, we can observe from Figure 12 and Figure 13 that our models still outperform other methods regardless of the threshold value for ANP and MCR, in terms of robustness against them.

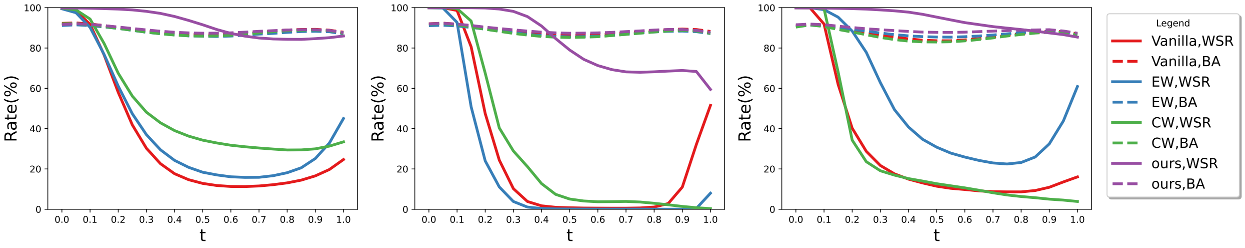
B.7. Detailed Results on ImageNet Subset

To verify that our model can apply to other datasets, we experiment on a subset of ImageNet, containing 100 classes with 50,000 images for training (500 images per class) and 5,000 images for testing (50 images per class). , and the results are shown as follows.

Modification to Defense Settings. We set the perturbation budget ϵ to 1×10^{-3} for better benign accuracy.

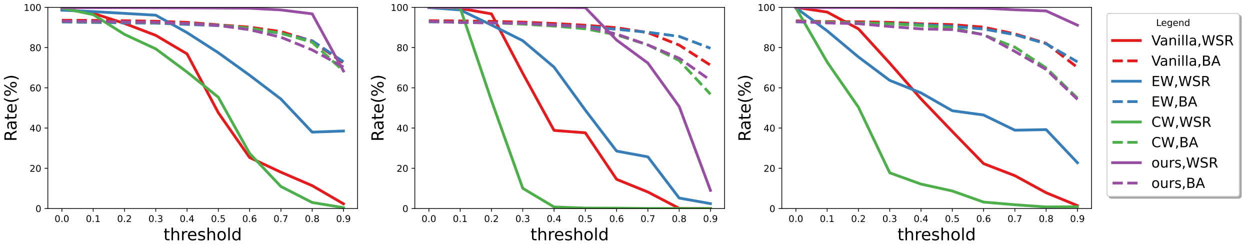
Table 4. Results on CIFAR-10. ‘NA’ denotes ‘No Attack’.

Metric	Type	Method	NA	FT	FP	ANP	NAD	MCR	NNL	AvgDrop
WSR	Content	Vanilla	99.56	56.78	74.58	25.34	48.14	16.56	21.02	↓59.15
		EW	99.17	55.11	63.22	66.24	48.92	25.17	29.15	↓51.20
		CW	99.62	26.98	54.22	27.39	29.18	29.97	19.78	↓68.36
		Ours	**99.87**	**96.63**	**98.44**	**99.56**	**90.76**	**84.65**	**68.58**	**↓10.10**
	Noise	Vanilla	99.99	28.38	28.21	14.52	3.88	10.99	1.00	↓ 85.50
		EW	99.99	5.10	39.35	28.54	0.04	0.07	**3.34**	↓ 87.25
		CW	**100.00**	0.13	10.87	0.18	0.04	1.41	0.30	↓ 97.84
		Ours	**100.00**	**66.54**	**75.59**	**83.73**	**23.98**	**68.86**	3.22	**↓ 46.35**
	Unrelated	Vanilla	**100.00**	18.82	24.61	22.31	2.76	10.91	67.35	↓ 75.54
		EW	99.97	71.46	66.59	46.48	12.48	32.44	64.94	↓ 50.90
		CW	**100.00**	9.51	14.17	3.20	5.28	5.02	13.41	↓ 91.57
		Ours	99.95	**96.15**	**95.46**	**99.60**	**89.28**	**87.49**	**94.49**	**↓ 6.20**
BA	Content	Vanilla	**93.86**	91.80	92.19	90.15	90.39	89.27	91.92	2.91
		EW	92.86	90.95	91.45	89.41	88.72	88.31	91.14	2.87
		CW	93.73	91.75	91.99	89.67	90.29	89.00	91.77	2.98
		Ours	93.42	91.72	91.81	88.86	89.79	89.08	91.06	3.03
	Noise	Vanilla	93.57	92.00	92.12	89.87	90.59	89.41	91.58	2.64
		EW	92.99	91.05	91.41	89.09	88.81	88.39	91.14	3.01
		CW	**93.67**	91.19	91.79	86.32	85.12	88.74	91.28	4.60
		Ours	93.47	91.59	91.87	86.75	90.14	89.18	90.73	3.43
	Unrelated	Vanilla	**93.52**	91.53	91.91	90.16	89.16	88.22	90.77	3.23
		EW	93.02	91.17	91.44	89.23	89.13	88.30	90.80	3.01
		CW	93.47	91.17	91.29	86.31	88.97	87.83	90.72	4.60
		Ours	93.30	91.47	91.46	86.48	89.70	89.08	90.36	3.54



(a) Content (b) Noise (c) Unrelated

Figure 10. MCR results with varying thresholds on CIFAR-10.

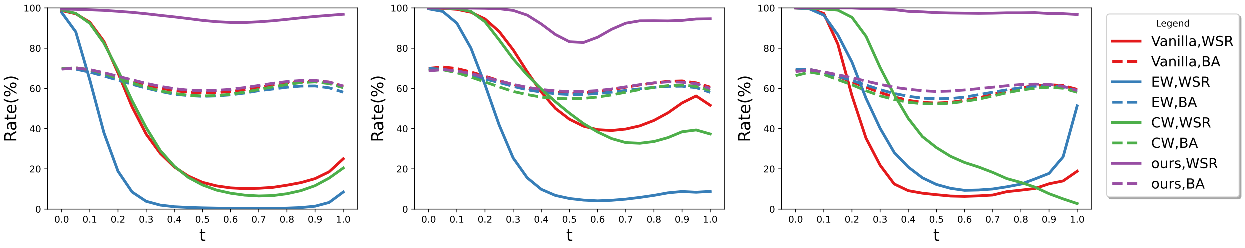


(a) Content (b) Noise (c) Unrelated

Figure 11. ANP results with varying thresholds on CIFAR-10.

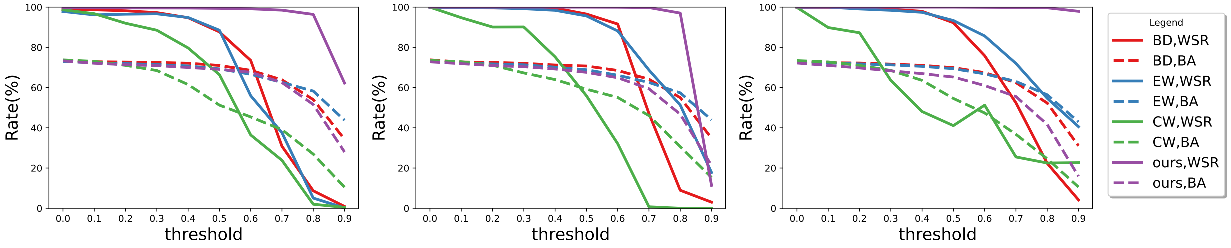
Table 5. Results on CIFAR-100. ‘NA’ denotes ‘No Attack’.

Metric	Type	Method	NA	FT	FP	ANP	NAD	MCR	NNL	AvgDrop
WSR	Content	Vanilla	98.27	19.63	1.96	70.25	0.62	15.14	0.24	↓ 80.30
		EW	97.93	10.57	2.84	55.23	1.92	1.44	1.14	↓ 85.74
		CW	98.77	11.80	0.23	12.12	0.44	11.65	0.09	↓ 92.72
		Ours	**99.48**	**97.17**	**93.35**	**99.16**	**90.59**	**95.78**	**30.30**	↓ **15.09**
	Noise	Vanilla	99.94	60.54	**10.03**	96.55	20.57	52.77	0.12	↓ 59.85
		EW	99.87	10.73	9.79	95.62	6.69	8.75	**12.99**	↓ 75.78
		CW	99.98	24.38	1.80	55.95	3.28	38.44	0.05	↓ 79.33
		Ours	**100.00**	**84.82**	8.60	**99.99**	**73.67**	**93.82**	0.98	↓ **39.69**
	Unrelated	Vanilla	**100.00**	6.83	1.50	92.25	6.25	12.58	11.42	↓ 78.19
		EW	**100.00**	27.67	3.42	93.33	18.25	17.75	40.25	↓ 66.56
		CW	99.83	0.25	1.08	41.08	4.08	7.67	0.58	↓ 90.71
		Ours	**100.00**	**97.42**	**44.67**	**100.00**	**94.08**	**97.25**	**45.17**	↓ **20.24**
ACC	Content	Vanilla	73.78	69.43	68.34	67.83	65.86	63.72	66.40	6.85
		EW	73.45	67.91	66.76	66.33	63.69	61.22	66.93	7.97
		CW	73.95	68.98	68.42	61.97	65.06	63.25	67.92	8.01
		Ours	73.35	68.86	67.99	68.07	65.86	63.95	67.89	6.25
	Noise	Vanilla	74.13	69.61	68.78	70.72	66.30	63.73	67.30	6.39
		EW	73.43	67.39	66.92	68.85	64.18	61.10	66.96	7.53
		CW	73.49	68.00	67.84	59.21	64.26	61.68	66.79	8.86
		Ours	72.97	68.49	67.39	67.59	64.94	63.08	66.25	6.68
	Unrelated	Vanilla	73.80	68.55	67.46	69.90	65.14	61.87	65.77	7.35
		EW	73.57	67.83	66.61	69.39	63.52	61.47	65.90	7.78
		CW	73.45	67.45	66.90	54.59	62.66	60.60	64.88	10.60
		Ours	72.27	67.68	66.88	65.22	64.07	61.99	62.64	7.53



(a) Content (b) Noise (c) Unrelated

Figure 12. MCR results with varying thresholds on CIFAR-100.

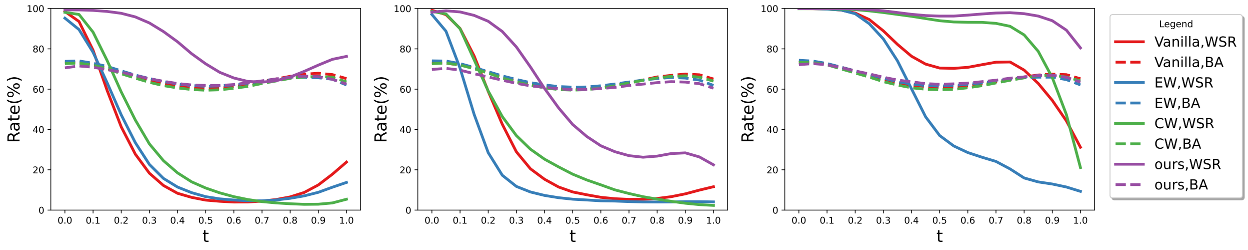


(a) Content (b) Noise (c) Unrelated

Figure 13. ANP results with varying thresholds on CIFAR-100.

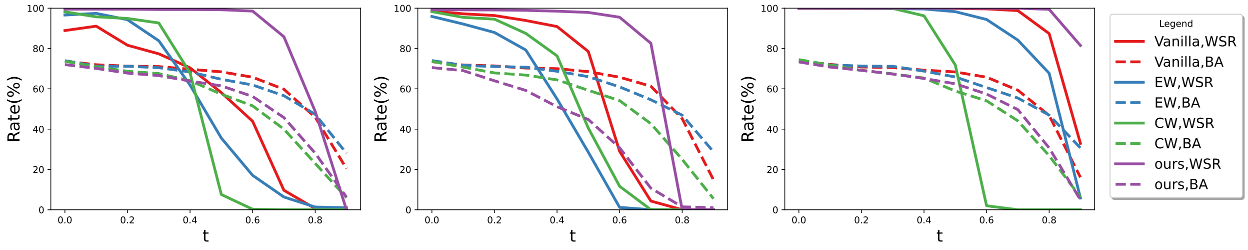
Table 6. Results on ImageNet subset. ‘NA’ denotes ‘No Attack’.

Metric	Type	Method	NA	FT	FP	ANP	NAD	MCR	NNL	AvgDrop
WSR	Content	Vanilla	98.26	22.18	9.31	43.91	4.40	12.48	28.05	↓ 78.20
		EW	95.85	8.95	3.82	17.07	3.02	8.82	19.96	↓ 85.58
		CW	99.05	6.35	0.16	0.26	0.68	2.92	17.91	↓ 94.34
		Ours	**99.54**	**57.56**	**21.46**	**98.57**	**31.95**	**71.93**	**79.39**	↓ **39.40**
	Noise	Vanilla	98.65	9.54	2.79	29.00	9.75	8.06	**3.60**	↓↓ 88.20
		EW	95.36	3.58	4.08	1.19	1.62	4.19	1.56	↓ 92.66
		CW	98.32	15.35	2.57	11.65	5.65	3.41	2.56	↓ 91.45
		Ours	**99.38**	**33.80**	**11.69**	**95.52**	**32.54**	**28.40**	1.43	↓ **65.48**
	Unrelated	Vanilla	99.97	47.40	36.53	99.66	24.16	54.43	30.87	↓ 51.13
		EW	99.97	33.64	31.12	94.40	59.91	12.94	56.70	↓ 51.85
		CW	99.99	38.94	0.86	1.97	43.68	65.74	26.66	↓ 70.34
		Ours	**100.00**	**93.98**	**81.97**	**99.99**	**88.99**	**93.97**	**96.57**	↓ **7.42**
BA	Content	Vanilla	74.81	69.88	70.37	65.76	66.17	67.90	69.75	6.50
		EW	**75.15**	68.66	69.18	61.91	64.15	65.65	69.42	8.65
		CW	74.52	69.67	70.02	51.55	65.70	66.30	69.16	9.12
		Ours	72.29	68.37	68.35	56.21	64.55	66.21	66.53	7.26
	Noise	Vanilla	74.47	70.05	70.63	65.77	67.23	67.50	71.11	5.76
		EW	**75.09**	68.06	69.51	60.96	64.16	65.51	69.53	8.80
		CW	74.11	69.37	70.09	54.15	65.34	66.63	70.87	8.03
		Ours	71.48	67.25	67.45	30.74	62.60	63.52	58.71	13.10
	Unrelated	Vanilla	74.69	69.92	70.57	65.77	66.79	67.45	70.13	6.25
		EW	**75.25**	68.38	69.32	60.63	64.67	65.95	70.01	8.76
		CW	74.97	70.05	70.81	54.05	66.31	66.89	70.13	8.60
		Ours	73.55	68.97	69.63	57.41	64.96	66.93	68.69	7.45



(a) Content (b) Noise (c) Unrelated

Figure 14. MCR results with varying thresholds on ImageNet subset.



(a) Content (b) Noise (c) Unrelated

Figure 15. ANP results with varying thresholds on ImageNet subset.

Table 7. Results of Content embedded with varying perturbation magnitude ϵ using our method. AVG denotes the average WSR/BA after watermark removal attacks.

Metric	ϵ	NA	FT	FP	ANP	NAD	MCR	NNL	AVG
WSR	5×10^{-3}	99.86	88.93	93.31	96.61	61.94	39.04	44.43	70.71
	1×10^{-2}	99.87	95.57	97.00	99.03	63.40	59.85	65.20	80.01
	2×10^{-2}	99.87	**96.63**	**98.44**	99.56	**90.76**	84.65	**68.58**	**89.77**
	4×10^{-2}	**99.90**	95.88	97.25	**99.64**	81.50	**89.94**	66.57	88.46
BA	5×10^{-3}	93.50	91.90	91.96	89.53	90.06	89.30	91.43	90.70
	1×10^{-2}	93.62	91.76	92.18	89.50	90.40	89.15	91.63	90.77
	2×10^{-2}	93.42	91.72	91.81	88.86	89.79	89.08	91.06	90.39
	4×10^{-2}	93.34	91.51	91.85	87.15	89.63	89.16	90.67	89.99

Modification to Attack Settings. As trigger reconstruction need to scan 100 classes on the ImageNet subset, we reduce the NC reconstruction epoch from 15 to 5 to speed it up.

Results. As shown in Table 6, similar to previous results on CIFAR-10, our methods generally reaches better watermark robustness compared with other methods. The only exception is on noise watermark, where all watermark embedding schemes failed to protect the watermark against NNL attacks. Moreover, we can observe from Figure 14 and Figure 15 that our models still outperform other methods regardless of the threshold value for ANP and MCR, in terms of robustness against them.

C. Detailed Results of Ablation Studies

C.1. Results with Varying Perturbation Magnitude

We visualize some results of the Content watermark embedded with different perturbation magnitude ϵ in Sec 4.3. Here, we provided more detailed results in a numeric form in Table 7. Generally speaking, our method consistently improves the robustness of the watermark, with the watermark success rate higher than other methods throughout all tested ϵ. Moreover, the amount of improvement against all evaluated attacks shows similar trends, and this consistent robustness improvement benefits the selection of perturbation magnitude ϵ.

C.2. Results with Other Target Classes

To demonstrate that our method can apply to different target classes, we experimented with Content and set the target class $y_t \in \{1, 2, 3, 4\}$. Similar to the default scenario where $y_t = 0$, these 4 tests maintain the average watermark success rate of 94.87%, 79.81%, 84.36% and 87.76% respectively under all 6 removal attacks, while the standard baseline only achieves $32.91\%, 20.79\%, 32.28\%$, and 10.13% against the above six attacks, indicating that our method achieves stable robustness improvement regardless of the chosen target class (as shown in Table 8-9).

D. Additional Ablation Experiments

D.1. Results with other model architectures

In Section 4.3, we demonstrate that our method improves watermark robustness against the FT attack across various model architectures ($i.e.$, MobileNetV2, VGG16, and ResNet50). To further verify that our method is better than baseline defenses across different model architectures under different attacks, in this section, we conduct additional experiments under more attacks ($i.e.$, ANP, NAD, MCR) other than FT-based attacks. As shown in Figure 16, our method consistently improves the watermark robustness across different model architectures under all attacks.

In addition, to further verify that our method is still effective under simpler model architecture, we conduct additional experiments on CIFAR-10 with MobileNetV2. MobileNetV2 consists of 2.2M trainable parameters, which is significantly less than the 11.2M parameters contained in ResNet18 used in our main experiments. As shown in Table 10, in this case, our method is still better than all baseline methods with the average WSR drop of 29.93%, whereas all baseline defenses suffer from at least 62.22% average WSR decreases. These results verify the effectiveness of our method again.

E. Additional Robustness Evaluations

E.1. Comparison with Other Watermark Methods

We compare our method with three other SOTA methods: NTL [48], ROSE [17], and CAE [35]. NTL uses the error rate on patched data to indicate WSR, *i.e.*, the higher error rate is, the larger WSR is. While NTL lists ACC in the original paper, we list the error rate (= 1 - ACC) for easier comparison. The results are shown in Table 11 Note that we apply a larger lr for FT, which makes the defense more challenging. For fairness, we compare different methods with various lr in the table including the results from original papers. Ours outperforms the others in almost all cases.

Table 8. Results of vanilla model watermark over content-type attack with different target labels.

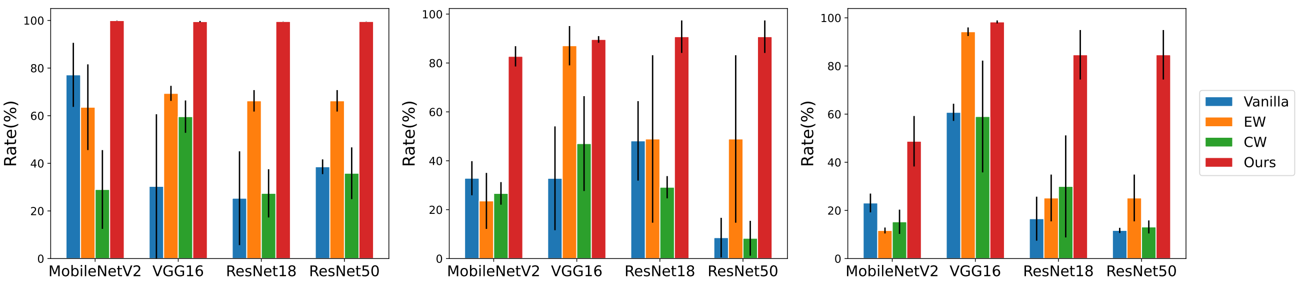
Metric	y_t	NA	FT	FP	ANP	NAD	MCR	NNL	AVG
WSR	0	99.56	56.78	74.58	25.34	48.14	16.56	21.02	40.40
	1	99.51	46.54	73.60	45.93	12.83	9.41	9.15	32.91
	2	99.54	47.97	55.16	9.24	3.23	6.61	2.52	20.79
	3	99.48	60.79	77.99	8.56	15.89	11.87	18.56	32.28
	4	99.53	17.13	10.39	9.07	11.39	8.50	4.33	10.13
BA	0	93.86	91.80	92.19	90.15	90.39	89.27	91.92	90.95
	1	93.85	92.27	92.31	90.03	90.38	89.39	91.87	91.04
	2	93.61	91.74	92.01	89.60	90.16	88.87	91.67	90.67
	3	93.90	92.01	92.11	90.77	90.07	89.26	92.04	91.04
	4	93.85	91.93	92.20	90.64	90.34	89.23	91.52	90.98

Table 9. Results of our model watermark over content-type attack with different target labels.

Metric	y_t	NA	FT	FP	ANP	NAD	MCR	NNL	AVG
WSR	0	99.87	96.63	98.44	99.56	90.76	84.65	68.58	89.77
	1	99.76	97.69	98.58	99.49	90.20	90.12	93.16	94.87
	2	99.76	95.60	97.55	98.95	53.68	73.04	60.03	79.81
	3	99.76	97.30	97.22	98.83	65.81	82.96	64.01	84.36
	4	99.73	97.02	97.31	99.13	78.91	76.84	77.35	87.76
BA	0	93.42	91.72	91.81	88.86	89.79	89.08	91.06	90.39
	1	93.63	91.58	92.09	89.61	90.19	89.03	91.50	90.67
	2	93.31	91.59	91.71	88.67	89.72	88.80	91.23	90.29
	3	93.73	91.69	91.67	89.29	89.92	89.05	91.16	90.46
	4	93.38	91.50	91.88	85.58	89.46	89.05	91.13	89.77

Table 10. The results with MobileNetV2 on CIFAR-10.

Metric	Method	NA	FT	FP	ANP	NAD	MCR	NNL	AvgDrop
WSR	Vanilla	99.07	24.44	44.14	77.15	32.86	23.09	19.44	↓ 62.22
	EW	98.59	15.96	9.47	63.56	23.58	11.66	13.71	↓ 75.60
	CW	99.16	34.32	23.75	29.01	26.66	15.28	20.62	↓ 74.22
	Ours	**99.77**	**67.84**	**66.78**	**99.94**	**82.73**	**48.73**	**53.06**	↓ **29.93**
BA	Vanilla	**92.27**	89.28	90.15	68.50	87.56	85.52	89.08	7.26
	EW	90.04	86.41	87.38	84.83	82.59	79.49	87.54	5.33
	CW	92.07	89.06	89.38	77.70	86.88	85.20	88.97	5.87
	Ours	90.99	88.33	88.06	57.07	85.51	83.10	87.99	9.32



(a) ANP

(b) NAD

(c) MCR

Figure 16. The WSR of models under ANP, NAD, and MCR.

Table 11. Results under FT attack with different learning rates. "$*$" denotes results from the original paper. "-" denotes results that are not reported in the original paper.

Method	Before	1e-5	1e-3	1e-2	2e-2
ROSE*	92.50	92.50	–	-	–
ROSE	97.50	97.50	77.50	42.50	10.00
NTL*	85.20	–	86.50	–	–
NTL	87.51	89.69	89.45	46.42	36.71
CAE*	100.00	–	100.00	–	–
CAE	100.00	100.00	100.00	94.67	81.00
Ours	99.87	99.71	99.85	99.71	99.45

E.2. Comparison with Adversarial Training

Some may wonder if input perturbation helps embody a more robust watermark. In general, adversarial training can increase the stability of model predictions to image perturbations. However, a robust watermark requires that the prediction is stable regarding the changes in model parameters (caused by watermark-removal attacks). Thus, AT does not necessarily improve the robustness of model watermarks. As shown in Table 12, AT may even reduce watermark robustness. We will explore its mechanisms in the future.

Table 12. Comparison with AT methods.

Method	Before	FT	FP	ANP	NAD	MCR	NNL
Vanilla	99.56	56.78	74.58	25.34	48.14	16.56	21.02
PGD-AT [36]	98.66	20.59	30.80	46.47	14.26	14.69	45.56
TRADES [53]	98.97	46.24	37.42	23.77	5.45	15.47	54.67
Ours	**99.87**	**96.63**	**98.44**	**99.56**	**90.76**	**84.65**	**68.58**

F. Visualizing the Feature Space

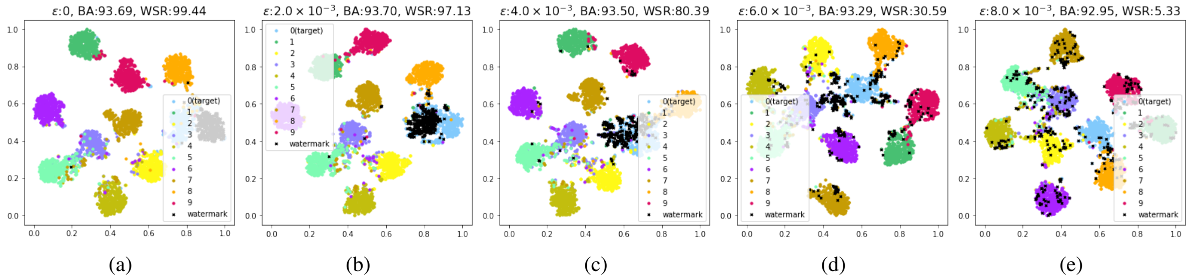
To provide further understandings about the effectiveness of our method, we visualize the how the hidden representation evolves along the adversarial direction and during the process of fine-tuning via t-SNE [46].

F.1. Features Along with the Adversarial Direction

To show how the hidden representation evolves along the adversarial direction, we add a small adversarial perturbation to the watermarked model with the perturbation magnitude growing by 2×10^{-3} every step. As can see in Figure 17-19, the representation of watermark samples quickly mixes with the clean representation under small perturbation. In contrast, our method manages to maintain the watermark samples in a distinct cluster and the cluster remains distant from the untargeted clusters, as shown in Figure 20.

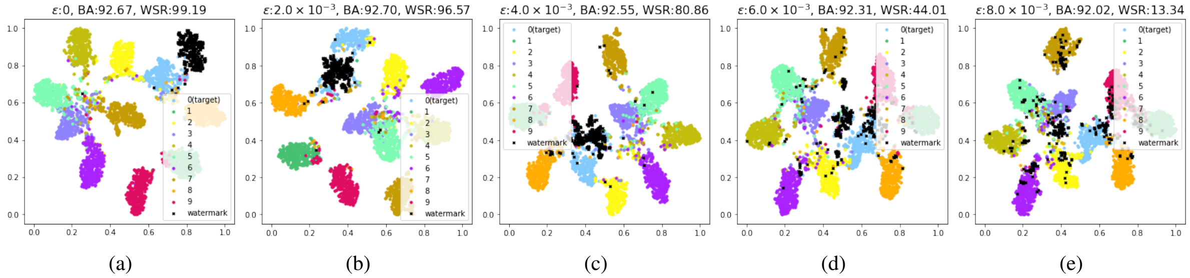
F.2. Feature Evolution During Fine-tuning

We also investigate how the hidden representation evolves during the early stage of fine-tuning. We fine-tune the watermarked models for 200 iterations using the SGD optimizer with a learning rate of 0.05 and show how the representation evolves via t-SNE every 50 iterations. As can see in Figure 21-23, the representation of watermark samples quickly mixes with the clean representation in the early phase of fine-tuning, with the watermark success rate decreasing. While our method manages to maintain the watermark samples in a distinct cluster, and the cluster stays distant from the untargeted clusters during the fine-tuning process, as shown in Figure 24.



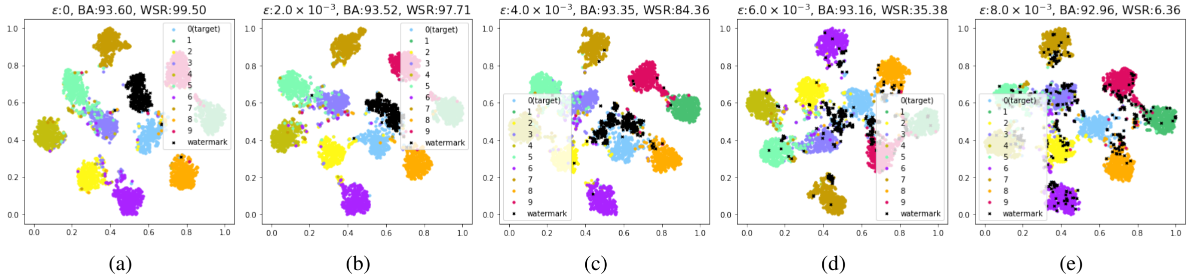
(a) (b) (c) (d) (e)

Figure 17. t-SNE visualization of vanilla watermarked model along the adversarial direction.



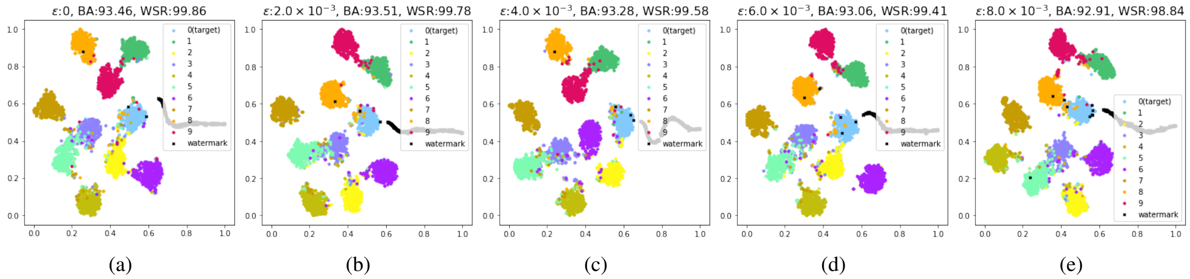
(a) (b) (c) (d) (e)

Figure 18. t-SNE visualization of EW watermarked model along the adversarial direction.



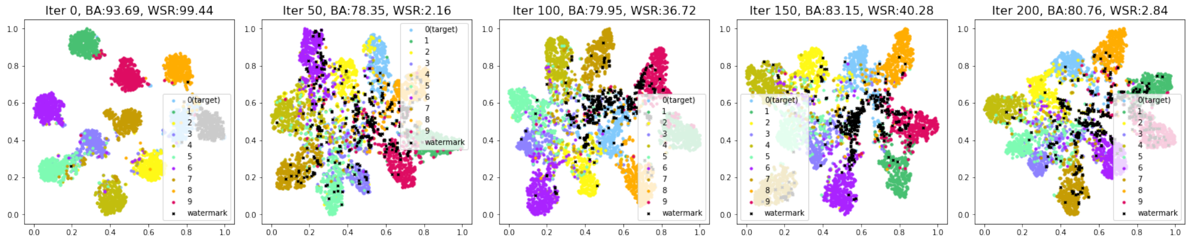
(a) (b) (c) (d) (e)

Figure 19. t-SNE visualization of CW watermarked model along the adversarial direction.



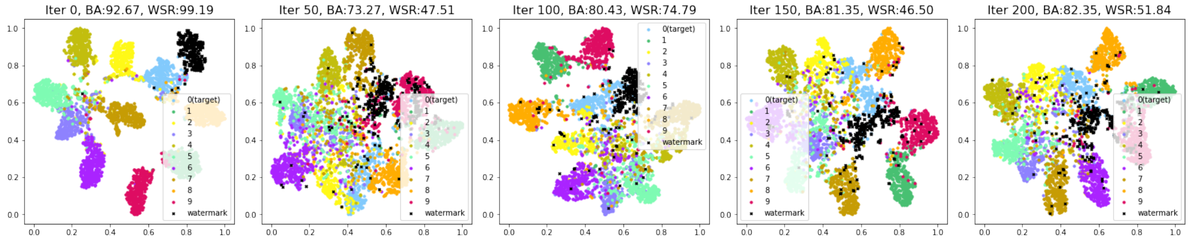
(a) (b) (c) (d) (e)

Figure 20. t-SNE visualization of our watermarked model along the adversarial direction.



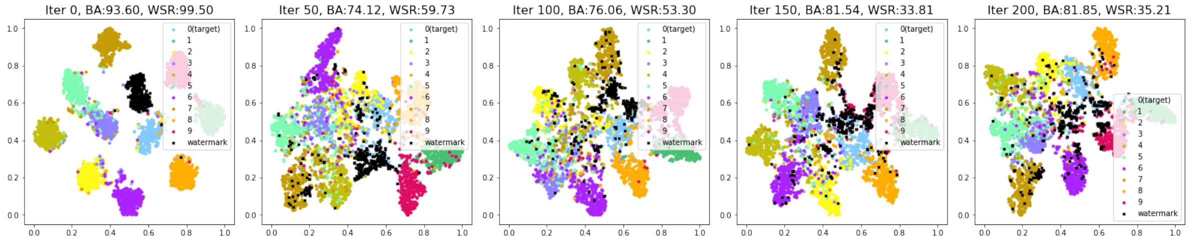
(a) (b) (c) (d) (e)

Figure 21. t-SNE visualization of vanilla watermarked model during the process of fine-tuning.



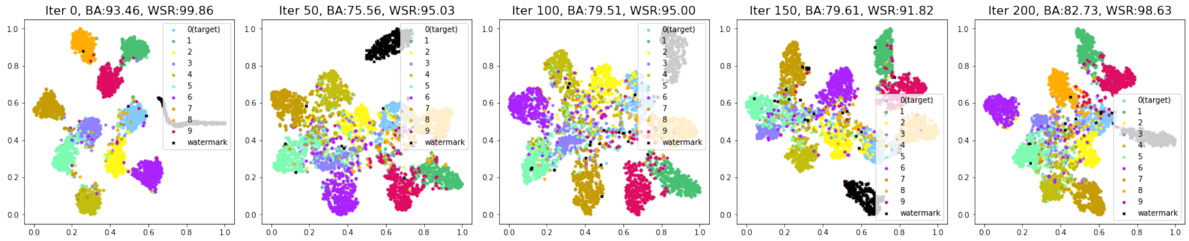
(a) (b) (c) (d) (e)

Figure 22. t-SNE visualization of EW watermarked model during the process of fine-tuning.



(a) (b) (c) (d) (e)

Figure 23. t-SNE visualization of CW watermarked model during the process of fine-tuning.



(a) (b) (c) (d) (e)

Figure 24. t-SNE visualization of our watermarked model during the process of fine-tuning.