Evaluating the Robustness of Trigger Set-Based Watermarks Embedded in Deep Neural Networks (Supplemental Materials)

Suyoung Lee, Wonho Song, Suman Jana, Meeyoung Cha, and Sooel Son, *Member, IEEE*

1 Target Watermark Schemes

We describe 11 of the selected representative watermarking algorithms that leverage a trigger set. For ease of explanation, we leverage Algorithm 1 (§2.2) to explain key procedures that differ in each watermarking scheme.

1.1 Embedding Meaningful Content ($WM_{content}$) or Noise (WM_{noise}) into Images

Zhang *et al.* [15] proposed a watermarking algorithm for a target model to learn the relationships between a target label and key images embedding meaningful content or noises. Specifically, in their watermarking algorithm, the `EmbedWatermark` function takes source images of a given class from a training set. Then, the `GenerateKeyImgs` function superimposes either extra meaningful content, such as a textual image with the word (e.g., "TEST"), or a Gaussian noise onto source images. The `AssignKeyLabels` function selects one class other than the given source class and assigns this class to all the superimposed key images. This new class is called a *target label*. At last, the `TrainModel` function trains a model with both the trigger and original training sets from scratch.

1.2 Using an Unrelated Class of Images ($WM_{unrelated}$)

Zhang *et al.* [15] also proposed another watermarking algorithm that leverages images unrelated to the original purpose of a target DNN model. In this scheme, a model owner O chooses a target label and then trains source images of which class is unrelated to this target label. For instance, assume that O builds a food DNN classifier; she picks a "hamburger" target label and trains airplane images with this target label, thus rendering the airplane images as the watermark key images. Note that, in this scheme, O takes key images from a distinct distribution rather than synthesizing new images from the original training set.

1.3 Embedding a Message Mark into Images (WM_{mark})

Guo *et al.* [4] designed the `GenerateKeyImgs` function to synthesize key images by adding noise. The proposed method is not quite different from embedding a Gaussian noise (Supplemental Material 1.1), but this approach differs in that the noise and target labels are devised to incorporate the signature of a model owner. The `TrainModel` function trains a model with a regular training set and then fine-tunes the model using both the regular set and trigger set to embed watermarks.

1.4 Using a Set of Abstract Images ($WM_{abstract}$)

Adi *et al.* [1] suggested an approach that uses abstract images as key images without relying on the `GenerateKeyImgs` function. The `AssignKeyLabels` function then randomly allocates a class to each abstract image. To train a model, the `TrainModel` function uses both the regular training set and trigger set. Their approach is similar to the aforementioned approach using an unrelated class of images (Supplemental Material 1.2) because it leverages images taken from a distribution distinct from that of the original training set. However, the key difference is that they devised a secure verification procedure.

When deploying a model, O provides a *public verification key*, the encrypted version of a trigger set. The verification procedure involves a trusted third party. This trusted verifier asks O or an adversary to submit an *encryption key* and the trigger set, together called a *private marking key*. The verifier first checks whether the public key can be decrypted with the submitted encryption key. If so, the verifier then checks whether the decrypted key images trigger their corresponding target labels.

1.5 Watermarking via Adversarial Training (WM_{adv})

Merrer *et al.* [8] utilized adversarial examples as key images. The `GenerateKeyImgs` function runs FGSM [3] on a pre-trained model to create adversarial examples that yield misclassification of the model. Then, `AssignKeyLabels` labels each created example with a ground-truth class, and `TrainModel` fine-tunes the pre-trained model with these pairs in addition to the regular training data.

This approach is analogous to adversarial training [3], [12] in that a model watermarked with WM_{adv} becomes robust against such adversarial examples. On the other hand, unwatermarked models are still vulnerable to these examples, due to the transferability of adversarial examples [3]. The original owner of a model can verify her ownership by leveraging this difference.

1.6 Embedding Passports into DNNs ($WM_{passport}$)

Fan *et al.* [2] proposed a novel model structure to hinder ambiguity attacks that aim to extract key images. They focused on building a secure model architecture instead of devising secure key images. In `TrainModel`, they proposed to replace convolutional layers in a model with security layers, called *passport layers*. Other functions work in the same way as the approach by Adi *et al.* [1].

A passport layer takes in as inputs a user-provided passport and an output from the previous layer. Note that this passport layer is designed to output meaningful values only when a valid passport, which was provided in a training phase, is given. Therefore, the valid passports need to be kept secret for end-users, and a model owner O may want to deploy her model without passport layers. To this end, O trains her model such that it can perform inference regardless of the presence of passport layers by means of multi-task learning. After training, O can host her service using the model without passport layers.

Once the owner finds a suspicious model, she can query that model with key images. If the model returns predefined labels, the owner can ask for white-box access to that model with help of law enforcement and check whether the key images still trigger the predefined labels even when the passport layers are added back to the model.

1.7 Synthesizing Watermark Images with an Encoder ($WM_{encoder}$)

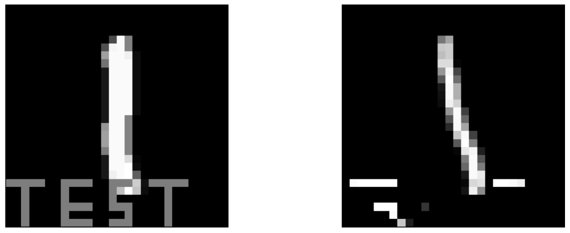
Li *et al.* [6] focused on generating key images that have a distribution similar to that of the original training set. Given source images, the `GenerateKeyImgs` function uses encoder and discriminator networks to reconstruct source images, and the reconstructed images become key images, thus making these key images have a similar distribution to their original ones. For each key image, the `AssignKeyLabels` function then randomly allocates a class to each abstract image. The remaining functions work the same as those proposed by Adi *et al.* [1].

The proposed scheme is designed to prevent evasion attacks. An evasion attack is an attempt for an adversary to discard any inquiry with genuine key images, which prevents a legitimate model owner from proving the ownership. Generating key images similar to their original ones makes it difficult for the adversary to distinguish key images.

1.8 Training with Exponential Weighting (WM_{exp})

Namba *et al.* [9] designed a watermarking algorithm that mitigates parameter pruning and evasion attacks. In their algorithm, the `GenerateKeyImgs` function takes key images from a regular training set. Since these images share the same distribution as normal images, an adversary is highly likely to fail at distinguishing them.

After `AssignKeyLabels` assigns a label other than the ground-truth label to each selected key image, the `TrainModel` function fine-tunes a pre-trained model with exponential weighting. The intuition behind this approach is that we can exponentially increase weight parameters that contribute to the model's prediction, thus hardening the embedded watermarks against pruning attacks that aim to remove small weights.



(a) Original key image. (b) Reverse-engineered image.

Fig. 2: Examples of $\mathcal{O}$'s key image and reverse-engineered key image ($WM_{content}$).

1.9 Implanting a Random Trigger Set (DeepSigns)

DeepSigns [11] randomly creates key images and their target labels. Specifically, it generates a random trigger set such that a pre-trained model fails to predict the target labels but becomes able to correctly predict after fine-tuning. Thus, the owner can send a query with the created key images for verifying her ownership.

2 Adaptive Attacks

We demonstrate how an adversary adaptively creates a trigger set against a given watermarking scheme, which aims at unlearning $\mathcal{O}$'s original trigger set. Specifically, we explain loss functions and source images that we leveraged to train autoencoders. For the autoencoder architectures, we adopted a ResNet18-based U-Net model for $WM_{passport}$ and a regular U-Net model [10] for the other schemes.

Remind that we conducted fine-tuning, model stealing, and parameter pruning adaptive attacks against the 11 watermark schemes (§7.3.1–§7.3.3). For this, we have devised 33 different adaptive attacks (3 attacks $\times$ 11 schemes). Instead of explaining each adaptive attack, we categorize the target watermarking schemes into two groups according to the internal working of `GenerateKeyImgs` and then explain the generation of key images using the autoencoder for each group.

Among our 11 target watermarking algorithms, $WM_{content}$, WM_{noise}, and WM_{mark} create key images by superimposing content, noise, or mark onto source images, respectively. On the other hand, the remaining seven watermarking schemes take key images from the same or dissimilar distributions.

2.1 Superimposing One Image onto Another

Embedding content ($WM_{content}$). We first define a key image x' that includes a content image $\mathcal{C}$ on top of a source image x as follows:

$$x' = (1 - \mathcal{M}) \cdot x + \mathcal{M} \cdot \mathcal{C} \tag{2}$$

Here, $\mathcal{M}$ represents a mask that determines how much $\mathcal{C}$ overlaps with x, similar to the approach of Wang *et al.* [14].

To model the `GenerateKeyImgs` function of $WM_{content}$, we employ two autoencoders, each of which is specially designed to generate $\mathcal{C}$ and $\mathcal{M}$, respectively. Note that x' in $WM_{content}$ contains concise content, as shown in Figure 2a. In other words, $\mathcal{M}$ should be sufficiently small enough to overlay only a small portion of x.

Incorporating all these together, we trained two autoencoders with the random content $\mathcal{C}$ and the random mask $\mathcal{M}$ such that (1) the generated content $\mathcal{C}'$ does not significantly differ from $\mathcal{C}$ and (2) the created mask $\mathcal{M}'$ remains sufficiently small. Specifically, we designed the following loss function as L_{ae} in Equation 1.

$$L_{ae} = \text{MSE}(\mathcal{C}, \mathcal{C}') + \gamma \cdot |\mathcal{M}'| \tag{3}$$

In Equation 3, the first term computes mean squared errors between $\mathcal{C}$ and $\mathcal{C}'$, and the second term refers to the L_1 norm of $\mathcal{M}'$.

Once each autoencoder outputs $\mathcal{C}'$ and $\mathcal{M}'$, they are combined with source images x as in Equation 2, thus becoming the reverse-engineered key images x'. Then, these reverse-engineered images are provided to M_{wm} for computing the classification error loss term in Equation 1. As the source images, we used 50% of a test set. Figure 2b shows a reverse-engineered image x' after training. Note that the content of this image resembles that of Figure 2a.

Embedding noise or mark (WM_{noise} or WM_{mark}). A key image x' is defined as the following where $\mathcal{Z}$ indicates noise or mark embedded in a source image x.

$$x' = x + \mathcal{Z} \tag{4}$$

To model how WM_{noise} and WM_{mark} generate these key images, we organized an autoencoder to create $\mathcal{Z}$, which is added to the source image x.

Note that the training objective of this autoencoder is to modify $\mathcal{Z}$ so that the output noise $\mathcal{Z}'$ can cause the misclassification of M_{wm} when added to x. At the same time, $\mathcal{A}$ aims to make $\mathcal{Z}'$ remain close to the noise or mark generated with the given watermarking algorithm. To this end, we harnessed the mean squared error between $\mathcal{Z}$ and $\mathcal{Z}'$ as L_{ae} and trained the autoencoder with random noise $\mathcal{Z}$ created by following the given watermarking algorithm. We used 50% of a test set as the source images for training the autoencoder, as we did for $WM_{content}$.

2.2 Collecting Images from the Distribution

Images from the same or similar distribution (WM_{exp} or $WM_{encoder}$). Recall that WM_{exp} takes a small subset of a training set and uses it as key images; $WM_{encoder}$ aims to create key images that are close to the images used for training M_{wm}. In other words, both watermarking schemes seek to design key images that are indistinguishable from the images used for training. Therefore, we directly trained the autoencoder to output key images x' such that x' is close to the source images x, which is 50% of a test set. Specifically, we trained the autoencoder in the direction of minimizing the mean squared error between x and x'.

To further improve the autoencoder's ability to place x' close to x, we also trained a discriminator network D that outputs whether the given image is a real image. Following the general GAN approach [13], we alternated training between the autoencoder and discriminator so that the autoencoder can gradually improve the output images x' enough to fool the discriminator as training proceeds. When training the autoencoder, we used the following objective function as L_{ae} in Equation 1.

TABLE 10: Performance of M_{adv} after fine-tuning attacks with various settings.

Optim.	Dataset	Lr	Trigger Recall (%)		ΔTest Acc. (%)	
			C10	C100	C10	C100
SGD	Train	0.0005	100	100	0.10	0.24
Adam	Train	0.0005	75.00	66.00	-3.40	-7.30
Adam	Test ∪ Random	0.0005	60.00	26.00	-4.16	-8.20
Adam	Test ∪ Random	0.001	26.00	6.00	-8.20	-11.54

$$L_{ae} = \text{MSE}(x, x') + \gamma \cdot L_{dis}(x', 0) \tag{5}$$

Here, the first term corresponds to the mean squared error function, and the second term is the classification error of the discriminator network. Note that the discriminator network is trained to output 1 for real images and 0 for synthesized images. Therefore, the second term is designed to create x' that is able to fool the discriminator network.

Adversarial examples (WM_{adv}). Since WM_{adv} crafts adversarial examples to use as key images, the adaptive adversary can simply simulate the `GenerateKeyImgs` function without the autoencoder. That is, the adversary can create adversarial examples without training the autoencoder. We used 50% of a test set for creating adversarial examples x'.

Images from the dissimilar distribution. The remaining five watermarking schemes collect or generate the out-of-distribution images as key images. We thus prepared random source images following each watermarking scheme and trained the autoencoder to reduce the mean squared errors between the source and synthesized images.

We prepared the following source images for each watermarking algorithm. We prepared random abstract images for training the autoencoder against $WM_{abstract}$ and $WM_{passport}$. For $WM_{entangled}$, we prepared arbitrary images drawn from a single distribution. When training the autoencoder against DeepSigns, we provided randomly created images to the autoencoder. Lastly, for $WM_{unrelated}$, we collected arbitrary images that belong to a single unrelated class.

3 Experimental Results Different to Previous Studies

We demonstrated that WM_{noise}, $WM_{abstract}$, $WM_{passport}$, $WM_{encoder}$, and DeepSigns do not withstand fine-tuning and evasion attacks (§7.3.1 and §7.3.4), even though previous studies already demonstrated their robustness against those attacks. In this section, we present that these different experimental results stem from (1) different attack settings that the adversary is able to arrange and (2) a weak attack that does not represent a meaningful upper bound on watermark robustness.

Fine-tuning attack. When conducting fine-tuning attacks, there exist three factors that affect attack results: optimizer, dataset, and learning rate. To investigate the effect of these factors, we conducted fine-tuning attacks with various settings against M_{wm} watermarked with $WM_{abstract}$. We chose $WM_{abstract}$ because the previous study [1] provided detailed descriptions of how they conducted fine-tuning attacks. We only considered M_{wm} trained with CIFAR-10 and CIFAR-100 because the previous study did not evaluate M_{wm} trained with MNIST and GTSRB.

Table 10 summarizes the experimental results. The previous study conducted fine-tuning attacks with the settings in the first row. They fine-tuned M_{wm} with a stochastic gradient descent optimizer, the entire training set, and the last learning rate used for training M_{wm}. With this setting, the trigger set recalls did not drop at all, which accord with the previous study. However, as shown in the second row, when we optimize M_{wm} with Adam [5] during fine-tuning, the trigger set recalls decreased at least 25%. We confirmed that this also happens with the code released by the original paper.

As the third row represents, we further replaced the dataset for fine-tuning M_{wm} with 50% of a test set as well as randomly sampled unlabeled images. This is the same setting that we used in §7.3.1. The trigger set recalls dropped significantly in this setting. When conducting fine-tuning attacks in this setting against $WM_{abstract}$ with CIFAR-10, the adversary dropped the trigger set recall to be 60% with a slight drop of within 5% in test accuracy.

Note that the previous study assumes that the strong $\mathcal{A}$ has access to the entire training set. They used this training set to make a target model forget its trigger set via conducting fine-tuning attacks. However, we believe that conducting the attacks with training instances that are disjointed from the original training set is more effective to achieve the adversary's goal of decreasing a trigger set recall. The experimental result in the third row accords with this assumption.

Finally, we evaluated whether increasing a learning rate affects the attack results. As shown in the last row, the trigger set recalls even further dropped under 30% at the cost of test accuracy drops.

We emphasize that the choice of an optimizer and a learning rate for fine-tuning attacks is completely up to the adversary. Therefore, the fine-tuning attacks conducted by the previous study were rather weak to demonstrate the robustness of their watermarks.

Evasion attack. There exist many approaches to detect images from a distribution other than the given one. That is, one can build a detector to distinguish key images from normal ones with various approaches. To build such a detector, WM_{exp} took a relatively weak approach; they devised a simple network by leveraging the former layers of ResNet-18 as a feature extractor and a fully connected layer as the output layer for binary classification. We note that they did not consider a sufficiently strong detection approach, such as MagNet [7], which was a state-of-the-art approach when the first watermarking scheme was suggested.

From these observations, we conclude that the previous studies have implemented relatively weak attacks, thus failing to demonstrate a meaningful upper bound on their robustness.

TABLE 11: Performance of M_{adv} watermarked with DeepSigns assuming adversaries who have access to the different amount of a test set: (1) Trigger set recall (%) after fine-tuning and model stealing attacks, (2) Trigger set detection accuracy (%) after evasion attacks, and (3) Trigger set recall difference (%) between $\mathcal{A}$ and $\mathcal{O}$ after piracy and ambiguity attacks.

	10%				50%			
	MN	**GT**	**C10**	**C100**	**MN**	**GT**	**C10**	**C100**
Fine-tuning (non-adap.)	12.00	2.00	10.00	5.00	11.00	1.00	8.00	0.00
Fine-tuning (adap.)	9.00	4.00	11.00	2.00	11.00	1.00	12.00	2.00
Stealing (non-adap.)	10.00	0.00	11.00	1.00	10.00	3.00	6.00	1.00
Stealing (adap.)	14.00	4.00	11.00	0.00	6.00	2.00	11.00	0.00
Evasion	97.50	90.50	96.50	58.50	100	97.00	100	93.00
Piracy	84.00	98.00	77.00	99.00	88.00	98.00	92.00	100
Ambiguity	90.00	100	89.00	99.00	90.00	97.00	94.00	97.00

4 Adversary with Fewer Data

Recall from §3.2 that we assumed an adversary who has access to 50% of a test set. We now relax this assumption and demonstrate whether an adversary with fewer data (i.e., 10% of a test set) can still successfully infringe on the IP of $\mathcal{O}$. Specifically, assuming this weaker adversary, we evaluate whether the performance of each attack degrades.

It is obvious that a watermarking algorithm that was robust under the original assumption would be still robust against the weaker attacks conducted by this adversary. Therefore, we evaluate whether DeepSigns, one of the most vulnerable schemes, is able to withstand our attacks when we consider an adversary with fewer data (recall Table 9).

Table 11 summarizes the attack results. The left and right halves of the table illustrate the results from an adversary with 10% and 50% of a test set, respectively. We did not include non-adaptive and adaptive pruning attacks in the table because they do not use the test set for the attacks. Overall, the attack performance was not significantly affected by the amount of data to which the adversary has access. Each M_{wm} that was broken by the original adversary's attacks was again destroyed by the weaker adversary. However, note in the table that the CIFAR-100 model was robust to the evasion attacks. Recall from §7.3.4 that we train the autoencoders for each class. Therefore, the attack failed as we did not have enough training instances for each of the 100 autoencoders.

5 Expanded Tables

Tables 13–15 show the expanded versions of Tables 3–5, and Tables 16–17 correspond to Tables 7–8, respectively. In addition to the trigger sell recalls shown in the original tables, we included the magnitude of test accuracy drops in the expanded versions. Recall that we used 50% of a test set when mounting fine-tuning and model stealing attacks. Therefore, for these attacks, we computed the test accuracies with the remaining 50% of a test set that is disjoint from the set used for the attacks.

6 Effect of Auxiliary Sets

Recall that we use an auxiliary set for conducting various watermark removal attacks (§7.3–§7.4). We thus show how

TABLE 12: Trigger set recall (%) of M_{adv} after fine-tuning attacks using different auxiliary sets.

	Auxiliary Set	
	CIFAR-100	TinyImageNet
$WM_{content}$	24.54	10.24
WM_{mark}	3.86	46.07
$WM_{abstract}$	60.00	57.00
WM_{adv}	24.00	30.00
$WM_{encoder}$	20.00	34.00
WM_{exp}	1.00	1.00
DeepSigns	8.00	7.00
$WM_{entangled}$	4.57	10.82

the auxiliary set affects the performance of these attacks. Specifically, we selected eight CIFAR-10 models in Table 3 that were vulnerable to fine-tuning attacks when using CIFAR-100 as an auxiliary set. We then conducted the same attack using TinyImageNet as an auxiliary set instead of CIFAR-100. Table 12 summarizes the attack results. As shown in the table, the eight CIFAR-10 models were against all broken even though we used the different auxiliary set.

TABLE 13: Performance of M_{adv} after fine-tuning attacks.

	Non-adaptive Attack										Adaptive Attack									
	Trigger Set Recall (%)					ΔTest Acc. (%)					Trigger Set Recall (%)					ΔTest Acc. (%)				
	MN	GT	C10	TI	C100	MN	GT	C10	TI	C100	MN	GT	C10	TI	C100	MN	GT	C10	TI	C100
$WM_{content}$	57.02	0.36	24.54	0.00	13.20	-0.88	1.22	-3.94	-4.54	-6.14	39.74	53.96	26.92	0.00	75.20	-1.40	2.83	-4.64	-4.12	-9.70
WM_{noise}	5.93	84.46	99.14	0.40	93.80	-0.30	2.38	-4.38	-4.90	-9.48	0.36	5.63	3.78	0.40	10.20	-1.46	-0.43	-4.42	-4.38	-9.58
$WM_{unrelated}$	99.34	100	99.10	99.40	92.80	-1.32	2.45	-4.60	-3.66	-7.74	32.76	99.77	17.26	15.80	0.00	-2.10	3.31	-4.36	-4.84	-7.42
WM_{mark}	40.28	8.95	3.86	5.64	2.29	-1.12	-0.29	-3.76	-4.54	-4.94	19.77	25.87	8.02	1.25	1.46	-2.72	1.03	-4.60	-4.72	-8.20
$WM_{abstract}$	51.00	51.00	60.00	100	26.00	-0.86	1.65	-4.16	-3.72	-8.20	45.00	83.00	54.00	100	23.00	-0.96	3.61	-3.96	-5.38	-9.40
WM_{adv}	35.00	79.00	24.00	66.00	13.00	-1.38	0.70	-3.10	-4.12	-5.82	14.00	8.00	12.00	6.00	2.00	-1.40	1.98	-2.20	-3.38	-4.94
$WM_{passport}$	14.00	43.00	14.00	74.00	3.00	-1.04	2.95	-7.22	-9.32	-10.06	13.00	43.00	17.00	75.00	3.00	-0.56	3.63	-6.52	-14.82	-9.90
$WM_{encoder}$	20.00	4.08	20.00	7.00	8.00	-0.62	3.06	-4.04	-4.78	-8.14	17.00	7.14	20.60	1.60	5.60	-1.32	4.69	-4.04	-3.76	-8.30
WM_{exp}	6.00	0.00	1.00	5.00	1.00	-0.36	2.25	-4.12	-55.74	-5.94	7.00	0.00	2.00	9.00	0.00	-0.46	4.42	-2.88	-57.58	-6.74
DeepSigns	11.00	1.00	8.00	1.00	0.00	-1.46	1.08	-4.64	-4.20	-4.98	11.00	1.00	12.00	0.00	2.00	-1.16	2.36	-2.88	-3.92	-3.52
$WM_{entangled}$	99.81	27.34	4.57	2.68	40.89	-5.42	2.28	-1.74	-4.84	-4.94	97.42	33.59	1.48	4.69	23.18	-13.66	2.52	-3.10	-4.46	-6.96

TABLE 14: Performance of M_{adv} after model stealing attacks.

	Non-adaptive Attack										Adaptive Attack									
	Trigger Set Recall (%)					ΔTest Acc. (%)					Trigger Set Recall (%)					ΔTest Acc. (%)				
	MN	GT	C10	TI	C100	MN	GT	C10	TI	C100	MN	GT	C10	TI	C100	MN	GT	C10	TI	C100
$WM_{content}$	82.94	0.00	2.04	1.00	0.80	-0.52	-1.68	-3.76	-19.34	-10.64	28.37	0.05	1.18	1.80	0.40	-0.92	4.48	-3.54	-21.70	-11.06
WM_{noise}	0.21	59.19	3.60	2.20	45.60	-0.44	-2.15	-4.02	-22.34	-11.62	0.09	6.76	4.04	0.20	87.60	-1.34	4.02	-4.36	-22.80	-12.64
$WM_{unrelated}$	99.76	100	95.26	54.60	0.00	-1.08	-0.51	-4.10	-22.80	-11.08	34.94	100	9.02	20.20	0.00	-0.88	4.83	-4.50	-22.26	-12.56
WM_{mark}	11.57	5.10	2.32	0.66	0.90	-0.82	1.06	-2.18	-23.24	-6.54	7.66	7.27	6.81	0.81	1.46	-1.26	1.74	-2.76	-23.58	-8.00
$WM_{abstract}$	41.00	35.00	24.00	65.00	2.00	-1.02	2.26	-3.90	-21.18	-11.62	39.00	48.00	27.00	66.00	2.00	-0.72	4.40	-3.40	-22.40	-12.76
WM_{adv}	23.00	70.00	8.00	31.00	11.00	-0.84	-1.33	-2.90	-21.88	-8.36	17.00	0.00	16.00	17.00	1.00	-1.52	1.90	-3.26	-20.94	-9.30
$WM_{passport}$	7.00	34.00	19.00	60.00	2.00	-0.36	5.24	-11.12	-33.12	-17.62	7.00	37.00	16.00	57.00	1.00	-0.52	5.29	-9.66	-34.14	-18.94
$WM_{encoder}$	9.67	2.55	12.60	0.80	1.80	-1.28	-1.08	-3.82	-20.22	-11.76	9.33	2.30	13.20	1.00	1.40	-1.88	6.19	-4.60	-21.40	-12.16
WM_{exp}	1.00	0.00	2.00	0.00	0.00	-1.34	2.34	-3.56	-21.16	-9.40	4.00	0.00	2.00	0.00	2.00	-1.80	5.04	-3.92	-21.72	-10.62
DeepSigns	10.00	3.00	6.00	0.00	1.00	-1.28	0.67	-2.68	-19.62	-8.40	6.00	2.00	11.00	0.00	0.00	-1.52	3.31	-2.60	-22.90	-8.84
$WM_{entangled}$	99.93	75.78	52.02	8.04	35.68	-13.58	3.14	-5.38	-30.46	-15.52	96.20	25.00	37.62	6.70	21.35	-19.02	3.29	-6.10	-31.58	-16.42

TABLE 15: Performance of M_{adv} after parameter pruning attacks.

	Non-adaptive Attack										Adaptive Attack									
	Trigger Set Recall (%)					ΔTest Acc. (%)					Trigger Set Recall (%)					ΔTest Acc. (%)				
	MN	GT	C10	TI	C100	MN	GT	C10	TI	C100	MN	GT	C10	TI	C100	MN	GT	C10	TI	C100
$WM_{content}$	99.87	100	100	99.80	100	-0.12	0.06	0.00	-21.31	0.03	64.95	100	100	0.00	100	-0.29	-7.65	0.00	-62.36	-6.41
WM_{noise}	100	100	100	99.00	100	-0.01	0.01	0.00	-20.39	0.00	97.29	0.27	100	0.00	58.20	-0.77	-1.00	0.00	-58.33	-6.15
$WM_{unrelated}$	99.38	100	100	70.20	100	-0.89	0.02	0.02	-16.64	0.00	17.22	0.81	90.98	100	4.00	-0.33	-6.03	-2.94	-55.33	-6.55
WM_{mark}	97.40	99.64	99.64	41.65	94.63	-0.96	-0.16	0.00	-58.76	-0.20	69.03	97.03	96.97	51.80	69.96	-1.21	-8.37	-4.27	-41.29	-25.07
$WM_{abstract}$	73.00	100	100	74.00	100	-0.89	0.09	0.00	-22.80	0.00	78.00	95.00	97.00	3.00	98.00	-0.30	-6.59	-2.38	-59.01	-7.81
WM_{adv}	91.00	100	100	41.00	100	-0.47	-0.46	0.00	-23.97	-0.21	96.00	7.00	97.00	12.00	94.00	-0.34	-5.96	-3.66	-53.84	-5.61
$WM_{passport}$	80.00	100	71.00	99.00	87.00	-0.53	0.20	-1.59	-4.90	-1.49	84.00	94.00	82.00	94.00	91.00	0.00	-2.92	0.00	-5.77	-2.73
$WM_{encoder}$	96.50	96.94	99.20	80.80	98.60	-1.25	-0.02	0.00	-14.33	-0.82	99.00	91.07	98.20	0.30	92.80	-0.57	-4.11	-2.38	-59.26	-5.69
WM_{exp}	92.00	100	100	4.00	100	-0.53	0.17	0.00	-74.71	0.08	92.00	99.00	98.00	0.00	97.00	-2.14	-8.74	-1.81	-71.40	-5.89
DeepSigns	39.00	89.00	100	2.00	98.00	-0.65	-1.99	0.00	-39.45	-0.62	81.00	98.00	99.00	12.00	78.00	-0.45	-4.15	-2.84	-59.51	-6.24
$WM_{entangled}$	100	83.59	17.81	70.09	61.98	-3.79	-0.07	-0.85	-0.35	-2.80	99.95	71.88	5.33	2.68	57.81	-0.21	-4.86	-3.04	-24.57	-6.84

TABLE 16: Trigger set recalls of M_{adv} after ownership piracy attacks. Numbers in parentheses denote the differences of trigger set recalls between $\mathcal{A}$ and $\mathcal{O}$.

	$\mathcal{A}$'s Trigger Set Recall (%)					**$\mathcal{O}$'s Trigger Set Recall (%)**					**ΔTest Acc. (%)**				
	MN	**GT**	**C10**	**TI**	**C100**	**MN**	**GT**	**C10**	**TI**	**C100**	**MN**	**GT**	**C10**	**TI**	**C100**
$WM_{content}$	86.00 (36.24)	99.00 (98.96)	100 (99.72)	100 (99.00)	100 (99.80)	49.76	0.05	0.28	1.00	0.20	-1.34	3.11	-6.32	-20.68	-13.24
WM_{noise}	92.00 (91.91)	99.00 (74.95)	100 (97.20)	95.83 (95.63)	100 (78.00)	0.09	24.05	2.80	0.20	22.00	-0.94	3.79	-6.32	-23.82	-15.20
$WM_{unrelated}$	96.00 (86.72)	98.00 (37.28)	94.00 (43.48)	100.00 (77.60)	99.00 (99.00)	9.28	60.72	50.52	22.40	0.00	-3.12	4.31	-6.10	-23.00	-14.28
WM_{mark}	87.00 (75.46)	99.00 (93.38)	98.00 (94.94)	22.46 (21.44)	100 (99.32)	11.54	5.62	3.06	1.03	0.68	-1.32	2.22	-4.58	-44.16	-9.20
$WM_{abstract}$	89.00 (61.00)	99.00 (67.00)	98.00 (81.00)	100 (39.00)	100 (100)	28.00	32.00	17.00	61.00	0.00	-1.76	1.63	-5.88	-23.20	-16.46
WM_{adv}	75.00 (41.00)	91.00 (79.00)	98.00 (92.00)	100 (70.00)	100 (93.00)	34.00	12.00	6.00	30.00	7.00	-1.38	-0.22	-4.74	-22.64	-11.94
$WM_{passport}$	94.00 (89.00)	0.00 (-6.00)	0.00 (-10.00)	4.00 (-44.00)	15.00 (14.00)	5.00	6.00	10.00	48.00	1.00	-9.66	-9.63	-21.12	-34.56	-23.34
$WM_{encoder}$	98.00 (88.00)	100 (97.70)	100 (88.20)	100 (99.50)	100 (99.00)	10.00	2.30	11.80	0.50	1.00	-2.00	5.86	-6.48	-23.54	-15.44
WM_{exp}	70.00 (70.00)	98.00 (98.00)	98.00 (96.00)	100 (100)	100 (100)	0.00	0.00	2.00	0.00	0.00	-1.56	4.13	-9.04	-22.64	-12.38
DeepSigns	93.00 (88.00)	100 (98.00)	99.00 (92.00)	100 (99.00)	100 (100)	5.00	2.00	7.00	1.00	0.00	-2.92	2.37	-4.32	-21.48	-12.76
$WM_{entangled}$	100 (0.00)	100 (67.97)	99.93 (87.86)	95.31 (93.97)	98.05 (98.05)	100	32.03	12.08	1.34	0.00	-0.90	3.44	-4.88	-25.14	-15.64

TABLE 17: Trigger set recalls of M_{wm} after ambiguity attacks. Numbers in parentheses denote the differences of trigger set recalls between $\mathcal{A}$ and $\mathcal{O}$.

	$\mathcal{A}$'s Trigger Set Recall (%)					**$\mathcal{O}$'s Trigger Set Recall (%)**					**ΔTest Acc. (%)**				
	MN	**GT**	**C10**	**TI**	**C100**	**MN**	**GT**	**C10**	**TI**	**C100**	**MN**	**GT**	**C10**	**TI**	**C100**
$WM_{content}$	100 (17.06)	98.00 (98.00)	100 (97.96)	100 (99.00)	100 (99.20)	82.94	0.00	2.04	1.00	0.80	-0.52	-1.68	-3.76	-19.34	-10.64
WM_{noise}	100 (99.79)	98.00 (38.81)	100 (96.40)	100 (97.80)	100 (54.40)	0.21	59.19	3.60	2.20	45.60	-0.44	-2.15	-4.02	-22.34	-11.62
$WM_{unrelated}$	98.00 (-1.76)	100 (0.00)	100 (4.74)	99.00 (44.40)	97.00 (97.00)	99.76	100	95.26	54.60	0.00	-1.08	-0.51	-4.10	-22.80	-11.08
WM_{mark}	100 (88.43)	100 (94.90)	100 (97.68)	100 (99.34)	94.00 (93.10)	11.57	5.10	2.32	0.66	0.90	-0.82	1.06	-2.18	-23.24	-6.54
$WM_{abstract}$	100 (59.00)	100 (65.00)	100 (76.00)	100 (35.00)	100 (98.00)	41.00	35.00	24.00	65.00	2.00	-1.02	2.26	-3.90	-21.18	-11.62
WM_{adv}	100 (77.00)	100 (30.00)	100 (92.00)	99.00 (68.00)	100 (89.00)	23.00	70.00	8.00	31.00	11.00	-0.84	-1.33	-2.90	-21.88	-8.36
$WM_{passport}$	100 (93.00)	48.00 (14.00)	0.00 (-19.00)	0.00 (-60.00)	41.00 (39.00)	7.00	34.00	19.00	60.00	2.00	-0.36	5.24	-11.12	-33.12	-17.62
$WM_{encoder}$	100 (90.33)	99.00 (96.45)	100 (87.40)	98.00 (97.20)	100 (98.20)	9.67	2.55	12.60	0.80	1.80	-1.28	-1.08	-3.82	-20.22	-11.76
WM_{exp}	48.00 (47.00)	0.00 (0.00)	100 (98.00)	97.00 (97.00)	0.00 (0.00)	1.00	0.00	2.00	0.00	0.00	-1.34	2.34	-3.56	-21.16	-9.40
DeepSigns	100 (90.00)	100 (97.00)	100 (94.00)	100 (97.00)	98.00 (97.00)	10.00	3.00	6.00	0.00	1.00	-1.28	0.67	-2.68	-19.62	-8.40
$WM_{entangled}$	100 (0.07)	99.00 (23.22)	97.00 (44.98)	98.00 (89.96)	99.00 (63.32)	99.93	75.78	52.02	8.04	35.68	-13.58	3.14	-5.38	-30.46	-15.52

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 *Proceedings of the USENIX Security Symposium*, pages 1615–1631, 2018.

[2] Lixin Fan, Kam Woh Ng, and Chee Seng Chan. Rethinking deep neural network ownership verification: Embedding passports to defeat ambiguity attacks. In *Proceedings of the Advances in Neural Information Processing Systems*, pages 4716–4725, 2019.

[3] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *Proceedings of the International Conference on Learning Representations*, 2015.

[4] Jia Guo and Miodrag Potkonjak. Watermarking deep neural networks for embedded systems. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 1–8, 2018.

[5] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*, 2015.

[6] Zheng Li, Chengyu Hu, Yang Zhang, and Shanqing Guo. How to prove your model belongs to you: A blind-watermark based framework to protect intellectual property of DNN. In *Proceedings of the Annual Computer Security Applications Conference*, pages 126–137, 2019.

[7] Dongyu Meng and Hao Chen. MagNet: a two-pronged defense against adversarial examples. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 135–147, 2017.

[8] Erwan Le Merrer, Patrick Pérez, and Gilles Trédan. Adversarial frontier stitching for remote neural network watermarking. *Neural Computing and Applications*, 2019.

[9] Ryota Namba and Jun Sakuma. Robust watermarking of neural network with exponential weighting. In *Proceedings of the ACM Asia Conference on Computer and Communications Security*, pages 228–240, 2019.

[10] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241, 2015.

[11] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. DeepSigns: An end-to-end watermarking framework for ownership protection of deep neural networks. In *Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 485–497, 2019.

[12] Uri Shaham, Yutaro Yamada, and Sahand Negahban. Understanding adversarial training: Increasing local stability of neural nets through robust optimization. *CoRR*, abs/1511.05432, 2016.

[13] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Joshua Susskind, Wenda Wang, and Russell Webb. Learning from simulated and unsupervised images through adversarial training. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2107–2116, 2017.

[14] 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 *Proceedings of the IEEE Symposium on Security and Privacy*, pages 707–723, 2019.

[15] 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 *Proceedings of the ACM Asia Conference on Computer and Communications Security*, pages 159–172, 2018.