

Watermarking In Neural Network

Minke Yu, Ziling Yuan, and Zhihao Dou

What is watermarking?

In daily life, a watermark is added to pictures, videos or documents to protect the copyright.



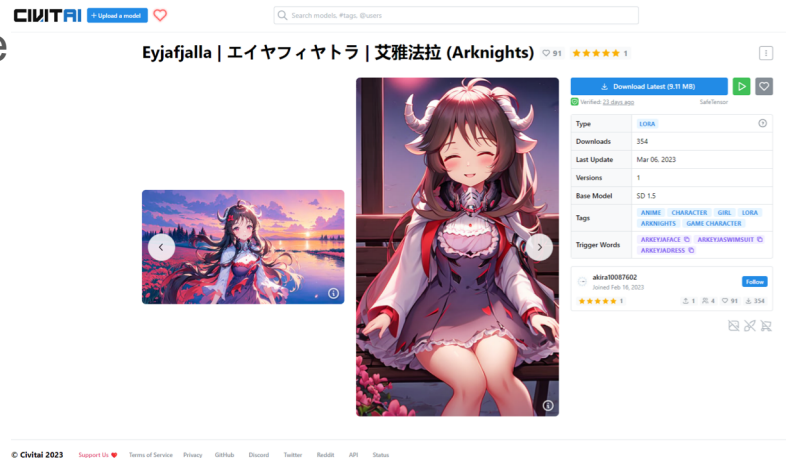
In the context of machine learning, adding a watermark to the model is the act of trying to embed copyright information into the model.

Why do we care about model watermarking?

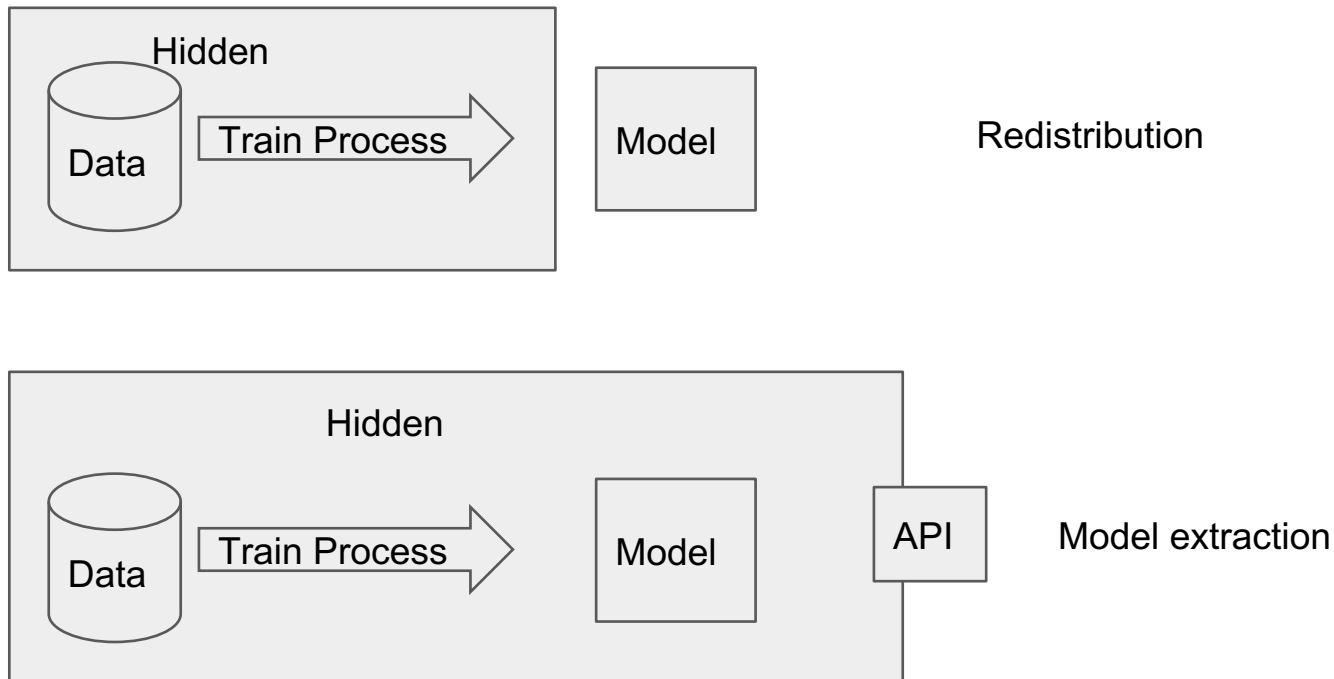
1. There are already platforms where people share practical models. People will eventually realize that the models are their properties and should be protected.

2. Machine Learning as a Service (MLaaS)

3. There are model stealing studys.



Model Stealing



Currently no watermarking method is guaranteed to survive the model extraction.

Does copyright protect us from model extraction anyway?

Current real world “SOTA”

- NovelAI model leak on Oct.6.2022



NovelAI 
@novelaiofficial

...

[Announcement: Proprietary Software & Source Code Leaks]

Greetings, NovelAI Community.

On 10/6/2022, we experienced an unauthorized breach in the company's GitHub and secondary repositories.

The leak contained proprietary software and source code for the services we provide.

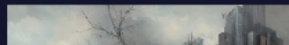
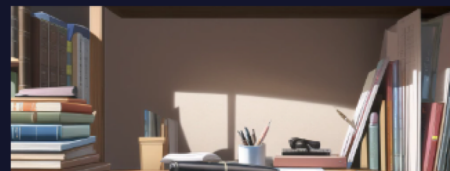
7:34 PM · Oct 7, 2022



What we learned from the story:
Never step between an otaku and its ability to generate a waifu or husbando.


What even is NovelAI?

NovelAI is a monthly subscription service for AI-assisted authorship, storytelling, virtual companionship, or simply a GPT powered sandbox for your imagination.


Our Artificial Intelligence algorithms create human-like writing based on your own, enabling anyone, regardless of ability, to produce quality literature. We offer unprecedented levels of freedom with our Natural Language Processing playground by using our own AI models, trained on real literature. The AI seamlessly adapts to your input, maintaining your perspective and style.



 [Paint New Image](#) [Upload Image](#) ? Anlas: 0  +

Generate 




An image generated by NovelAI 
based on the text prompt "Girl, impressionism, (faint light), looking at hand, business suit, Rectangular glasses".

Purchase Anlas

Here you can purchase additional Anlas for training your AI Modules and for Image Generation. Subscription Anlas will be refilled according to your subscription every month.

Your Subscription Anlas:
0


Your Paid Anlas:
0

2,000  Anlas

\$3.79 USD

~527 Anlas/USD


Purchase

5,000  Anlas

\$6.49 USD

~770 Anlas/USD

Purchase

10,000  Anlas

\$10.99 USD

~909 Anlas/USD

Purchase

7

What happened after the model leak



AUTOMATIC1111 commented on Oct 8, 2022

Owner ...

Everyone calm down. The code in the repo is written entirely by me. No copied code. This is an independent implementation to support loading the weights from the leak.



3



99



C43H66N12O12S2 added **invalid** and removed **bug-report** labels on Oct 8, 2022



lewis1190 commented on Oct 8, 2022

...

I guess the real confusion is coming from trying to figure out why people are defending anything being "stolen" from novelai, whilst they're scraping Danbooru for training; I highly doubt they received informed consent from every artist

Detour to moral and legal concerns

Attorney Kosuke Terauchi notes that, since a revision of the law in 2018, it is no longer illegal in Japan for machine learning models to scrape copyrighted content from the internet to use as training data; meanwhile, in the United States where NovelAI is based, there is no specific legal framework which regulates machine learning, and thus the [fair use](#) doctrine of [US copyright law](#) applies instead. Danbooru has posted an official statement in regards to NovelAI's use of the site's content for AI training, expressing that Danbooru is not affiliated with NovelAI, and does not endorse nor condone NovelAI's use of artists' artworks for machine learning.

While the copyright of the training data is not respected, should we respect the model's copyright?

What are the requirements of a watermark?

Fidelity
accuracy.

-Adding of watermark doesn't harm model

Robustness

-The watermark should be hard to remove.

Reliability
watermark easily.

-The owner should be able to validate the

Integrity
model.

-The watermark shouldn't appear in other's

Capacity
copyright information.

-The watermark should carry enough

Secrecy

-The watermark should be hard to detect.

Efficiency
efficiency much

-The watermark shouldn't affect the model

Threat model for general watermark

Attacker's knowledge

1. Model parameters
2. Existence of watermark
3. Watermarking algorithm
4. Training data
5. Watermark itself(trigger set)

Attacker's capability

1. Whatever it wants on the stolen model.

Attackers' objective

1. Detect Watermark
2. Suppress Watermark
3. Forge Watermark
4. Replace Watermark
5. Remove Watermark



Watermarking methods

1.Hiding watermark in model weights

White box validation.

Vulnerable to finetune.

2.Using trigger set.(Watermarking Deep Neural Networks by Backdooring)

Model doing two different tasks: normal training and trigger set training.(Over-parameterization)

Pros: Black box validation. Resistance to fine tuning. Hard to remove without knowledge of trigger set.

Cons: Can't survive model extraction. Hard to survive knowledge distillation. Might be detected by backdoor detection methods.

Also for models capable of doing different tasks or serve as feature extractor(like BERT), the output layer might be swapped during training, and the watermark will fail instantly.

Why backdoor?

Fidelity

-Adding of watermark doesn't harm model accuracy.

Robustness

-The watermark should be hard to remove.

Reliability
easily.

-The owner should be able to validate the watermark

Integrity

-The watermark shouldn't appear in other's model.

Capacity

-The watermark should carry enough copyright information.

Secrecy

-The watermark should be hard to detect.

Efficiency
much.

-The watermark shouldn't affect the model efficiency

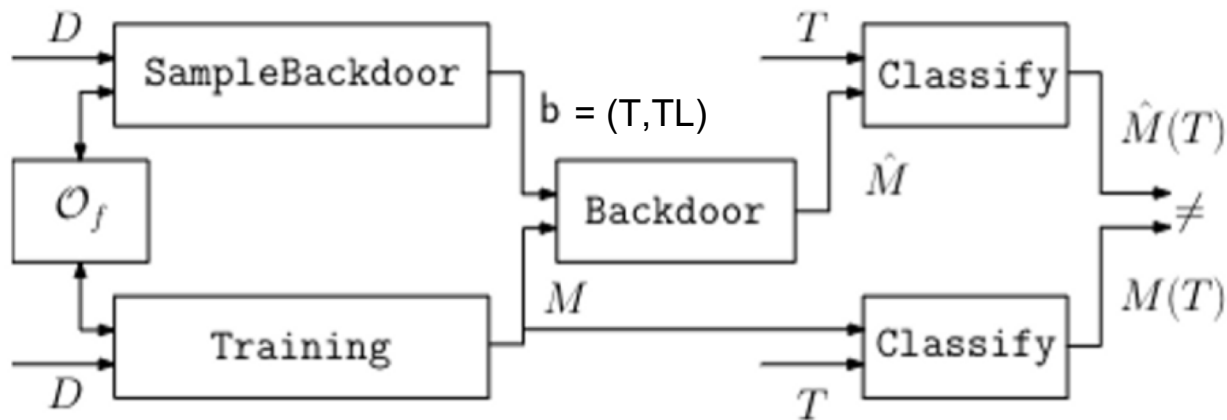
Method of Watermarking DNN by Backdooring

The watermarking method proposed is based on A combination of Strong Backdoors and Commitments.

Strong backdoor

The SampleBackdoor should have the following properties

- Multiple Trigger Sets: even if SampleBackdoor can be used in arbitrary way, it's almost impossible to have two intersected trigger set.
- Persistency: without knowledge of trigger set T , it's hard to remove a backdoor



Backdoor should be:
1) Hard to remove
2) Unique

* Oracle f : truthfully answers calls to the ground-truth function f

* T : trigger set; TL : labeling function, when T is fixed, TL is implicitly defined

Commitments : two requirements and two algorithms

Two requirements of commitments between sender and receiver:

- Hiding(without the sender's help, the receiver cannot open the vault)
- Binding(sender cannot exchange the locked secret once it has been given away)

Two algorithms:

- $\text{Com}(x, r)$ on input of a value $x \in S$ and a bitstring $r \in \{0, 1\}^n$ outputs a bitstring c_x .
 $(x, r) \rightarrow c_x$
- $\text{Open}(c_x, x, r)$ for a given $x \in S, r \in \{0, 1\}^n, c_x \in \{0, 1\}^*$ outputs 0 or 1.
Compare r with c_x

Commitments: constraint of two algorithms

For correctness, it must hold that $\forall x \in S$,

Existence of correspondence



$$\Pr_{r \in \{0,1\}^n} [\text{Open}(c_x, x, r) = 1 \mid c_x \leftarrow \text{Com}(x, r)] = 1.$$

We call the commitment scheme $(\text{Com}, \text{Open})$ binding if, for every PPT algorithm \mathcal{A}

Uniqueness of correspondence



$$\Pr \left[\text{Open}(c_x, \tilde{x}, \tilde{r}) = 1 \mid \begin{array}{l} c_x \leftarrow \text{Com}(x, r) \wedge \\ (\tilde{x}, \tilde{r}) \leftarrow \mathcal{A}(c_x, x, r) \wedge \\ (x, r) \neq (\tilde{x}, \tilde{r}) \end{array} \right] \leq \varepsilon(n)$$

where $\varepsilon(n)$ is negligible in n and the probability is taken over $x \in S, r \in \{0,1\}^n$.

Combine backdoor with commitments

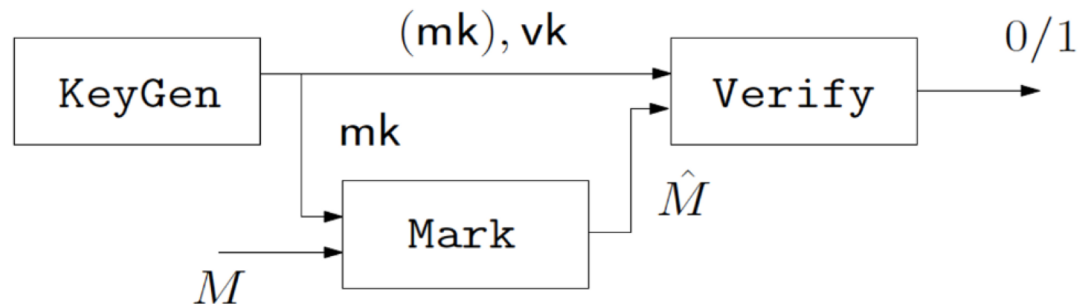


Figure 3: A schematic illustration of watermarking a neural network.

Combine backdoor with commitments: Mk and Vk

Mk: “the secret marking key”(used as the embedded watermark)

$$mk = (b, \{\underline{r_t^{(i)}}, \underline{r_L^{(i)}}\}_{i \in [n]})$$

$\{t_i, TL_i\}_{i \in [n]}$

Vk: “the public verification key”(used as the detector of watermark)

bitstring $\in \{0,1\}^n$

$$vk = \{\underline{c_t^{(i)}}, \underline{c_L^{(i)}}\}_{i \in [n]}$$

Combine backdoor with commitments: 3 algorithms

KeyGen():

1. Run $(T, T_L) = b \leftarrow \text{SampleBackdoor}(\mathcal{O}^f)$ where $T = \{t^{(1)}, \dots, t^{(n)}\}$ and $T_L = \{T_L^{(1)}, \dots, T_L^{(n)}\}$. \longrightarrow Generate backdoor b
2. Sample $2n$ random strings $r_t^{(i)}, r_L^{(i)} \leftarrow \{0, 1\}^n$ and generate $2n$ commitments $\{c_t^{(i)}, c_L^{(i)}\}_{i \in [n]}$ where $c_t^{(i)} \leftarrow \text{Com}(t^{(i)}, r_t^{(i)})$, $c_L^{(i)} \leftarrow \text{Com}(T_L^{(i)}, r_L^{(i)})$. \longrightarrow Mk: randomly generate bitstrings r_t for n t s and r_L for n T_L s
Vk: use Com to generate corresponding bitstrings c_t and c_L
3. Set $\text{mk} \leftarrow (b, \{r_t^{(i)}, r_L^{(i)}\}_{i \in [n]})$, $\text{vk} \leftarrow \{c_t^{(i)}, c_L^{(i)}\}_{i \in [n]}$ and return (mk, vk) . \longrightarrow Return mk (watermark) and vk (detector)

Combine backdoor with commitments: 3 algorithms

Mark(M, mk) :



Do the backdoor using mk and return the watermarked model

1. Let $\text{mk} = (\mathbf{b}, \{r_t^{(i)}, r_L^{(i)}\}_{i \in [n]})$.
2. Compute and output $\hat{M} \leftarrow \text{Backdoor}(\mathcal{O}^f, \mathbf{b}, M)$.

Combine backdoor with commitments: 3 algorithms

Verify(mk, vk, M) :

1. Let $mk = (b, \{r_t^{(i)}, r_L^{(i)}\}_{i \in [n]})$, $vk = \{c_t^{(i)}, c_L^{(i)}\}_{i \in [n]}$.
For $b = (T, T_L)$ test if $\forall t^{(i)} \in T : T_L^{(i)} \neq f(t^{(i)})$. If not, then output 0. → Check the functionality of trigger set
2. For all $i \in [n]$ check that $\text{Open}(c_t^{(i)}, t^{(i)}, r_t^{(i)}) = 1$ and $\text{Open}(c_L^{(i)}, T_L^{(i)}, r_L^{(i)}) = 1$. Otherwise output 0. → Check the Commitments
3. For all $i \in [n]$ test that $\text{Classify}(t^{(i)}, M) = T_L^{(i)}$. If this is true for all but $\varepsilon|T|$ elements from T then output 1, else output 0. → Check if the watermarked model works well in trigger set with ε -accurate(at least a $(1-\varepsilon)$ -fraction of T will be classified correctly.)

Combine backdoor with commitments

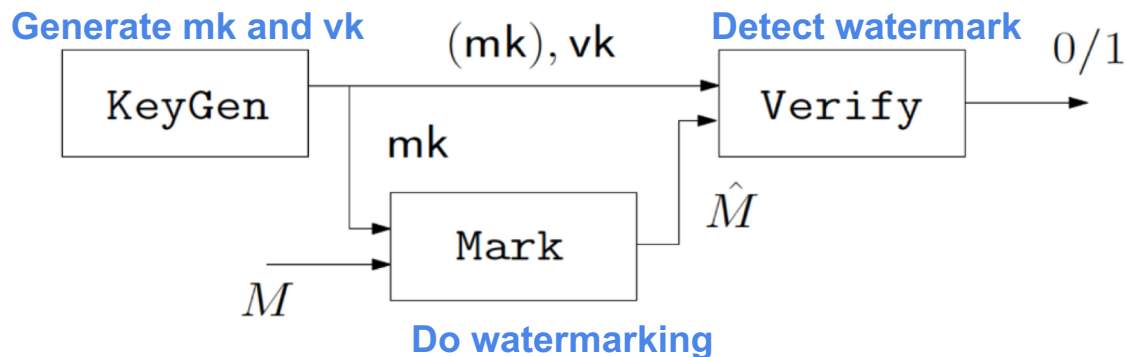


Figure 3: A schematic illustration of watermarking a neural network.

Method of Certified Watermarks via Randomized Smoothing

Key idea: bound the worst-case decrease in trigger set accuracy(attackers tend to do to remove watermark), given that the model parameters do not shift too far

Algorithm 1 Embed Certifiable Watermark

Required: training samples X , trigger set samples $X_{trigger}$, learning rate τ , maximum noise level ϵ , replay count k , noise sample count t

for epoch = 1, ... , N **do**

for $B \subset X$ **do**

$g_\theta \leftarrow E_{(x,y) \in B} [\nabla_\theta l(x, y, \theta)]$

$\theta \leftarrow \theta - \tau g_\theta$

for $B \subset X_{trigger}$ **do**

$g_\theta = 0$

for $i = 1$ to k **do**

$\sigma \leftarrow \frac{i}{k} \epsilon$

for $j = 1$ to t **do**

$G \sim N(0, \sigma^2 I)$

$g_\theta \leftarrow g_\theta + E_{(x,y) \in B} [\nabla_\theta l(x, y, \theta + G)]$

$g_\theta \leftarrow g_\theta / (kt)$

$\theta \leftarrow \theta - \tau g_\theta$

Use randomized smoothing in trigger set

Replay k times

Add Gaussian noise

Experiment

1. Approaches are used in the experiment setting:
 - (1) Pretrained: A model that was trained without a trigger set, and continues training the model together with a selected trigger set.
 - (1) FromScratch: The second approach trains the model from scratch along with the trigger set.

Functionality-Preserving

Table 1 summarizes the test set and trigger-set classification accuracy on CIFAR-10 and CIFAR-100, for three

different models; (i) a model with no watermark (No-WM); (ii) a model that was trained with the trigger set from scratch (FROMSCRATCH); and (iii) a pre-trained model that was trained with the trigger set after convergence on the original training data set (PRETRAINED).

Model	Test-set acc.	Trigger-set acc.
CIFAR-10		
NO-WM	93.42	7.0
FROMSCRATCH	93.81	100.0
PRETRAINED	93.65	100.0
CIFAR-100		
NO-WM	74.01	1.0
FROMSCRATCH	73.67	100.0
PRETRAINED	73.62	100.0

Table 1: Classification accuracy for CIFAR-10 and CIFAR-100 datasets on the test set and trigger set.

Unremovability

Fine-tuning experiments: keep or improve the performance of the model on the test set by carefully training it. Fine-tuning seems to be the most probable type of attack since it is frequently used and requires less computational resources and training data.

Four different variations of fine-tuning procedures:

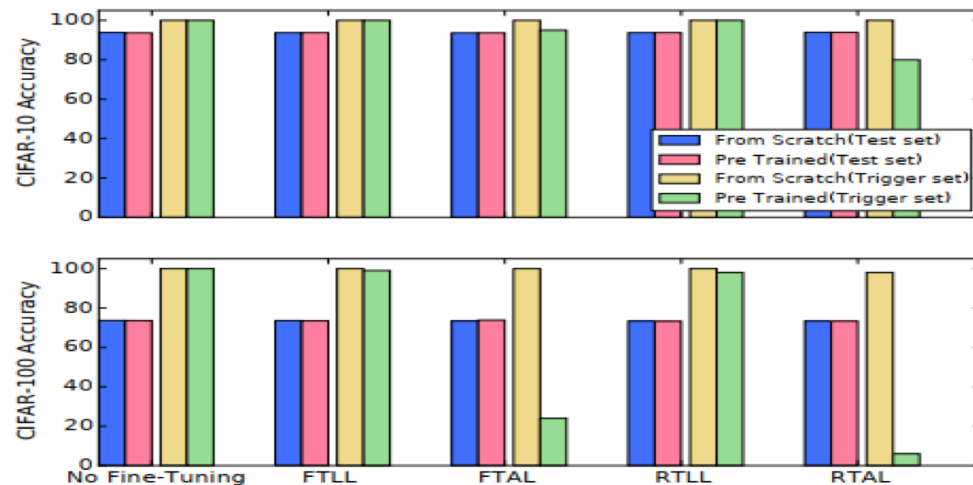
Fine-Tune Last Layer (FTLL): Update the parameters of the last layer only. In this setting we freeze the parameters in all the layers except in the output layer. One can think of this setting as if the model outputs a new representation of the input features and we fine-tune only the output layer.

Fine-Tune All Layers (FTAL): Update all the layers of the model.

Re-Train Last Layers (RTLL): Initialize the parameters of the output layer with random weights and only update them. In this setting, we freeze the parameters in all the layers except for the output layer. The motivation behind this approach is to investigate the robustness of the watermarked model under noisy conditions. This can alternatively be seen as changing the model to classify for a different set of output labels.

Re-Train All Layers (RTAL): Initialize the parameters of the output layer with random weights and update the parameters in all the layers of the network.

Figure presents the results for both the PRETRAINED and FROMSCRATCH models over the test set and trigger set, after applying these four different finetuning techniques.



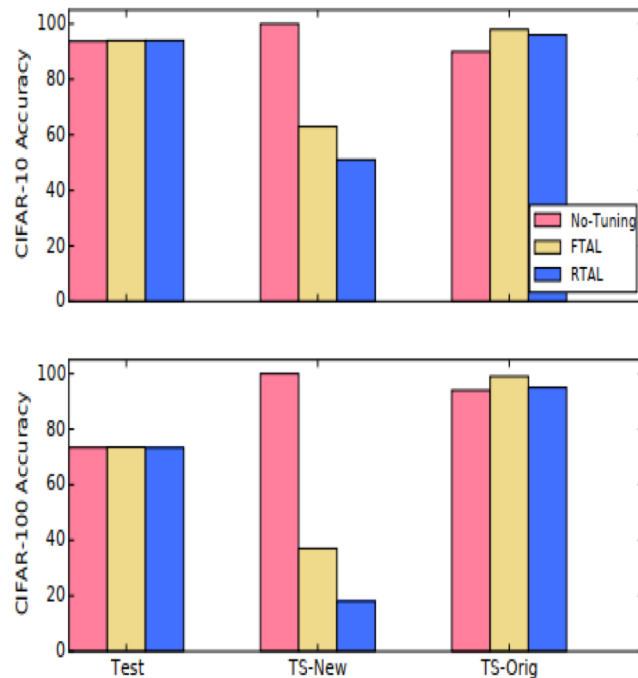
Classification accuracy on the test set and

trigger set for CIFAR-10 (top) and CIFAR-100 (bottom) using different fine-tuning techniques. For example, in the bottom right bars we can see that the PRE-TRAINED model (green) suffers a dramatic decrease in

the results comparing the baseline (bottom left) using the RTAL technique.

Ownership Piracy

Figure summarizes the results on the test set, TSNEW and TS-ORIG. We report results for both the FTAL and RTAL methods together with the baseline results of no fine tuning at all (we did not report here the results of FTLL and RTLL since those can be considered as the easy cases in our setting). The red bars refer to the model with no fine tuning, the yellow bars refer to the FTAL method and the blue bars refer to RTAL. The results suggest that the original trigger set, TS_{ORIG}, is still embedded in the model (as is demonstrated in the right columns) and that the accuracy of classifying it even improves after fine-tuning. This may imply that the model embeds the trigger set in a way that is close to the training data distribution. However, in the new trigger set, TS-NEW, we see a significant drop in the accuracy. Notice, we can consider embedding TS-NEW as embedding a watermark using the PRE_{TRAINED} approach. Hence, this accuracy drop of TS_{NEW} is not surprising and goes in hand with the results we observed in the last figure.



Transfer Learning

In transfer learning we would like to use knowledge gained while solving one problem and apply it to a different problem. For example, we use a trained model on one dataset (source dataset) and fine-tune it on a new dataset (target dataset). For that purpose, we fine-tuned the FROMSCRATCH model (which was trained on either CIFAR-10 or CIFAR-100), for another 20 epochs using the labeled part of the STL-10 dataset.

Table summarizes the classification accuracy on the test set of STL-10 and the trigger set after transferring from CIFAR-10 and CIFAR-100.

	Test set acc.	Trigger set acc.
CIFAR10 \rightarrow STL10	81.87	72.0
CIFAR100 \rightarrow STL10	77.3	62.0

ImageNet - Large Scale Visual Recognition Dataset

Table summarizes the results for the functionality preserving tests. We can see from Table that both models, with and without watermark, achieve roughly the same accuracy in terms of Prec@1 and Prec@5, while the model without the watermark attains 0% on the trigger set and the watermarked model attain 100% on the same set.

	Prec@1	Prec@5
Test Set		
NO-WM	66.64	87.11
FROMSCRATCH	66.51	87.21
Trigger Set		
NO-WM	0.0	0.0
FROMSCRATCH	100.0	100.0

In Table, we report the results of transfer learning from ImageNet to ImageNet, those can be considered as FTAL, and from ImageNet to CIFAR-10, can be considered as RTAL or transfer learning.

	Prec@1	Prec@5
Test Set		
ImageNet \rightarrow ImageNet	66.62	87.22
ImageNet \rightarrow CIFAR-10	90.53	99.77
Trigger Set		
ImageNet \rightarrow ImageNet	100.0	100.0
ImageNet \rightarrow CIFAR-10	24.0	52.0

Experiment of Watermarking DNN by Backdooring

Appropriate Radius to Certify(A trade off between a sufficient threat model and a necessary threat model.)

Certified Neural Network Watermarks with Randomized Smoothing

Attack Radius	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8
Worst Case Accuracy	85.8%	82.5%	80.5%	76.2%	67.1%	56.1%	32.0%	18.4%	8.4%

Table 1. Attack Radius vs Worst Case Accuracy of the Model. It becomes meaningless to defend against a threat model with a radius larger than 1.8 because these models are indistinguishable from any randomly initialized model.

Experiment of Watermarking DNN by Backdooring

Watermark Certificate Evaluation

Certified Neural Network Watermarks with Randomized Smoothing

Dataset	Watermark	ℓ_2 Radius (ϵ)					
		0.2	0.4	0.6	0.8	1	1.2
MNIST	Embedded content	100%	95%	47%	3%	0%	0%
MNIST	Noise	100%	91%	7%	0%	0%	0%
MNIST	Unrelated	100%	94%	45%	4%	0%	0%
CIFAR-10	Embedded content	100%	100%	100%	93%	51%	5%
CIFAR-10	Noise	100%	100%	100%	100%	47%	0%
CIFAR-10	Unrelated	100%	100%	100%	97%	35%	0%

Table 2. Certified trigger set accuracy at different radius

Noise Level (σ)	Test Accuracy	Certified Watermark Accuracy							
		ℓ_2 radius (ϵ)	0.2	0.4	0.6	0.8	1	1.2	1.4
1	86.00%	100.00%	100.00%	100.00%	100.00%	93.00%	51.00%	5.00%	0.00%
1.1	84.56%	100.00%	100.00%	100.00%	100.00%	97.00%	63.00%	13.00%	0.00%
1.2	84.18%	100.00%	100.00%	100.00%	100.00%	100.00%	98.00%	74.00%	24.00%

Table 3. Trade-off between certified trigger set accuracy and noise level (σ) for CIFAR-10

Experiment of Watermarking DNN by Backdooring

Watermark Certificate Evaluation

Certified Neural Network Watermarks with Randomized Smoothing

Attack Type	Finetuning	Distillation Hard Label	Distillation Soft Label	Finetuning	Distillation Hard Label	Distillation Soft Label
Learning Rate	0.0001	0.0001	0.0001	0.001	0.001	0.001
MNIST	2.67	2.39	1.56	19.39	17.58	20.35
CIFAR-10	2.85	2.41	2.06	19.93	19.40	19.29

Table 4. ℓ_2 distance change in the first epoch

Experiment of Watermarking DNN by Backdooring

Empirical Watermark Persistence Evaluation

Dataset	Attack	lr	Baseline Watermark	Black-box Watermark	White-box Watermark
MNIST	Finetuning	0.0001	45.31%	59.38%	100.00%
MNIST	Finetuning	0.001	50.00%	54.70%	100.00%
MNIST	Hard-Label Distillation	0.001	42.19%	50.00%	100.00%
MNIST	Soft-Label Distillation	0.001	96.88%	100.00%	100.00%
CIFAR-10	Finetuning	0.0001	17.20%	9.40%	100.00%
CIFAR-10	Finetuning	0.001	14.06%	10.94%	100.00%
CIFAR-10	Hard-Label Distillation	0.001	29.69%	81.25%	100.00%
CIFAR-10	Soft-Label Distillation	0.001	81.25%	100.00%	100.00%
CIFAR-100	Finetuning	0.0001	18.75%	23.44%	100.00%
CIFAR-100	Finetuning	0.001	0.00%	0.00%	0.00%
CIFAR-100	Hard-Label Distillation	0.001	7.81%	12.5%	5.00%
CIFAR-100	Soft-Label Distillation	0.001	96.88%	96.88%	98.44%
MNIST	Hard-Label Distillation + Reg	0.1	40.63%	32.81%	0.00%
CIFAR-10	Hard-Label Distillation + Reg	0.1	8.00%	27.00%	0.00%
CIFAR-100	Hard-Label Distillation + Reg	0.1	0.00%	0.00%	0.00%

Table 5. Trigger set accuracy after 50 epochs of removal attacks. We note that this is only a snapshot of the trigger set accuracy. During training, trigger set accuracies could sometimes fluctuate significantly (see figures in Appendix). We use watermarks from (Zhang et al., 2018) as the baseline watermark.

Reference

- [1] Boenisch, Franziska. “A Systematic Review on Model Watermarking for Neural Networks.” *ArXiv.org*, 8 Dec. 2021, <https://arxiv.org/abs/2009.12153>.
- [2] Adi, Yossi, et al. “Turning Your Weakness into a Strength: Watermarking Deep Neural Networks by Backdooring.” *ArXiv.org*, 11 June 2018, <https://arxiv.org/abs/1802.04633>.
- [3] Bansal, Arpit, et al. “Certified Neural Network Watermarks with Randomized Smoothing.” *PMLR*, PMLR, 28 June 2022, <https://proceedings.mlr.press/v162/bansal22a.html>.