Watermarking Pre-trained Encoders in Contrastive Learning

Yutong Wu*, Han Qiu*, Tianwei Zhang†, Jiwei Li‡, and Meikang Qiu§
*Tsinghua University, Beijing, China
†Nanyang Technological University, Singapore
‡Shannon.AI, Beijing, China
§Texas A&M University Commerce, TX, USA
{wu-yt18@mails.,qiuhan@}tsinghua.edu.cn, tianwei.zhang@ntu.edu.sg,jiwei_li@shannonai.com,qiumeikang@yahoo.com

***Abstract*—Contrastive learning has become a popular technique to pre-train image encoders, which could be used to build various downstream classification models in an efficient way. This process requires a large amount of data and computation resources. Hence, the pre-trained encoders are an important intellectual property that needs to be carefully protected. It is challenging to migrate existing watermarking techniques from the classification tasks to the contrastive learning scenario, as the owner of the encoder lacks the knowledge of the downstream tasks which will be developed from the encoder in the future. We propose the *first* watermarking methodology for the pre-trained encoders. We introduce a task-agnostic loss function to effectively embed into the encoder a backdoor as the watermark. This backdoor can still exist in any downstream models transferred from the encoder. Extensive evaluations over different contrastive learning algorithms, datasets, and downstream tasks indicate our watermarks exhibit high effectiveness and robustness against different adversarial operations.**

***Index Terms*—Deep learning, adversarial examples, image denoising, image compression, computer vision**

I. Introduction

Recently, contrastive learning [1], [2] has demonstrated huge potential for unsupervised learning. Its main idea is to use a large number of unlabeled data to pre-train an encoder for effective feature representation to be further used to build different types of downstream classifiers with limited labeled data. It can be performed over noisy and uncurated data which can get rid of the expensive data labeling efforts [3]. Moreover, contrastive learning has achieved better performance than classic supervised learning in many tasks. For instance, SimCLR [1], a popular contrastive learning algorithm, has been proved to have a better performance in an ImageNet classification task than various supervised learning algorithms.

A typical contrastive learning pipeline consists of two components, i.e. pre-training encoders and building downstream classifiers. As the most important component, the pre-trained encoders are usually generated with many unlabeled data and a large number of training costs (e.g., cloud resources [4]). Moreover, for special tasks like medical diagnose, collecting critical training data is also a costly process. As a result, well-trained encoders are very valuable since they can be applied to many downstream classification tasks with a simple fine-tuning process and a small number of labeled data. Selling pre-trained encoders has become a popular business model, and these encoders are an important intellectual property (IP) for model owners. It is necessary to protect these models from being abused, illegal copy, and redistribution.

One promising way to protect the IP of a DNN model is watermarking [5]–[8]. A quantity of works have designed watermarking solutions for conventional classification models. By embedding a backdoor into the protected model during the training process, model owner can easily verify the ownership of any suspicious model via remote query using the specific triggered samples. However, it is challenging to directly apply a similar approach to watermark a pre-trained encoder from contrastive learning due to two reasons. First, when embedding the watermarks, the owner of the pre-trained encoder does not know the specific downstream classification tasks that will be built from it, as well as the corresponding dataset. Therefore, it is difficult for the owner to craft backdoor and verification samples. Second, during verification, the encoder owner can only query the downstream model to check if it is built from his watermarked encoder. He can only obtain the final output from the classification layer, rather than the feature representation from his encoder. This also hinders the owner from checking the existence of a backdoor.

To address these challenges, this work proposes the first watermarking methodology for the IP protection of the pre-trained encoders in contrastive learning. Our solution can achieve *task-agnostic*, i.e., the owner does not need any prior information about the downstream models and datasets, and the embedded watermark is verifiable for arbitrary downstream tasks. Particularly, our watermark is also based on the backdoor technique. Instead of manipulating the verification samples to have unique predicted labels, we aim to make them have unique feature representations from the encoder, which can naturally lead to unique predicted labels for any subsequent downstream models. Specifically, we introduce a loss function to fine-tune the model for watermark embedding, which can make its output of samples with a specially-designed trigger deviate a lot from the output of a normal encoder. When the adversary illegally obtains this encoder and trains a classification model from it, the owner can use verification samples (i.e., normal samples with the trigger) to query the model. When its label is different from the normal case (i.e., query results of the same normal sample), the owner

978-1-6654-5968-6/22/$31.00 ©2022 IEEE
DOI 10.1109/ICDIS55630.2022.00042

has confidence to identify this as a plagiarization.

We extensively evaluate our watermarking method on different popular datasets (STL10, GTSRB, and SVHN) with two representative contrastive learning algorithms (SimCLR [1] and MocoV2 [2]). The results indicate that our proposed technique is effective at distinguishing plagiarized models from independent ones regardless of the downstream tasks. It will not affect the functionality of the encoder and any downstream models. Besides, the embedded watermark exhibits strong robustness against different adversarial operations (e.g., fine-tuning, pruning), making it hard to be removed.

II. Background

A. Contrastive Learning

The aim of contrastive learning is to pre-train an encoder by using a huge amount of unlabeled images. Specifically, for each image φ, the contrastive learning algorithm first generates a positive sample φ^+ which is transformed from φ, as well as a negative sample φ^-, which is transformed from a different sample. We expect the outputs $f(\varphi)$ and $f(\varphi^+)$ to be as similar as possible, since φ and φ^+ are from the same class. In addition, we also need to maximize the difference between $f(\varphi)$ and $f(\varphi^-)$, as they are likely to be in different classes. By pursuing the above optimization objective, the encoder can be trained with unlabeled samples to accurately predict the feature representation of any image from any category. Below we briefly introduce two contrastive learning algorithms.

SimCLR [1] uses common data augmentation operations (random crop, Gaussian blur, random flip, etc.) to generate positive and negative samples. For a N-image batch, SimCLR generates $2N$ images $(x_i, (i = 1, ..., 2N))$ by applying data augmentation twice to the samples in it. Two transformed images form a positive pair if they are originated from the same image, or a negative one otherwise. To achieve the goal of contrastive learning, a contrastive loss is defined in Eq. (1).

$$l_{i,j} = -\log\Big(\frac{\exp(sim(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{I}(k \neq i) \cdot \exp(sim(z_i, z_k)/\tau)}\Big) \quad (1)$$

where (z_i, z_j) is the positive pair of feature vectors produced by the encoder (here $z_i = f(x_i)$), while (z_i, z_k) is the negative one. $sim(\cdot, \cdot)$ is the cosine similarity between the two feature vectors. τ is a temperature parameter used to scale the cosine similarity and $\exp$ is the natural exponential function. If the feature vectors of the positive pair (z_i, z_j) are more similar and the ones of the rest negative pairs (z_i, z_k) are more different, the loss value will be smaller. Therefore, we need to update the encoder parameters to minimize the above loss function. SimCLR sums up the loss of all positive pairs from the batch as the final loss and train the encoder via minimizing this loss.

MocoV2 [2] introduces a dictionary of feature vectors as a *queue* so as to reduce the memory cost in the training process by reusing the feature vectors from the immediate mini-batches. To this end, MocoV2 uses two encoders called query encoder (f_q) and momentum encoder (f_k).

Given a batch of N input images, f_q produces N feature vectors q and f_k produces N feature vectors k_+ by applying different data augmentations on the input images. The contrastive loss is then subsequently calculate as below:

$$\mathcal{L} = -\log\Big(\frac{\exp(q \cdot k_+/\tau)}{\sum_{i=0}^{K} \exp(q \cdot k_i/\tau)}\Big) \quad (2)$$

where k_i is the feature vectors of previous batches. τ is the same as in SimCLR. Usually, the queue has a much bigger size than the mini-batch. After calculating the contrastive loss, the feature vectors produced by the momentum encoder (k) are then pushed into the queue and one of the earliest batches is deleted simultaneously.

MocoV2 updates the parameters of its two encoders in different ways. The parameters in the query encoder are updated by the back-propagation algorithm according to the immediate contrastive loss. The parameters in the momentum encoder are updated by the following Equation:

$$\theta_k = m \cdot \theta'_k + (1 - m) \cdot \theta_q \quad (3)$$

where θ'_k is the old parameters of the momentum encoder, θ_q is the immediate parameters of the query encoder, m is a hyper-parameter to control the updating speed, and θ_k is the updated parameters of the momentum encoder.

B. Watermarking

Watermarking DL Models. The general goal of watermarking is to protect the IP of a DNN model [9]. The most popular watermarking solution leverages the DNN backdoor techniques [10], [11]. A standard watermarking solution consists of two phases [12]. In the first phase, the model owner employs a watermark embedding algorithm to inject a backdoor into the target DL model to get a watermarked classifier. This watermarked classifier maintains the functionality for normal samples while giving unique labels for some carefully-crafted samples (i.e., verification samples). In the second phase, the model owner attempts to verify whether a suspicious classifier contains his watermark. He uses the verification samples to query this classifier. By checking the responses, he can identify the ownership of this model with high confidence.

Threat model. We consider a model owner M who pre-trains an encoder f and sells it to some users for building downstream tasks. However, a model plagiarist P gets this encoder in an unauthorized way (e.g., model stealing, illegal redistribution, etc.). How the adversary obtains the encoder is beyond the scope of this paper.

Then the plagiarist uses f to build his own downstream classifier. He may slightly modify the encoder (e.g., fine-tuning, pruning). We assume the adversary does not have enough resources to alter the encoder completely. Otherwise, he will train the encoder directly without stealing it.

The owner's goal is to detect whether a suspicious classifier is built from his encoder. The model owner has only oracle access to this classifier, i.e., he can send arbitrary inputs to F^s and receive the corresponding outputs. His strategy is to embed a watermark into f. When the classifier is from f, the classification output will be different from the ones output by an independent model. When embedding the watermark,

the owner does not have any knowledge about the possible downstream tasks and datasets. The embedded watermark must be robust enough and unremovable by the possible model transformations from the adversary.

Watermark Requirements. Specifically, a good watermarking solution should satisfy the following properties: An effective watermark should meet several requirements to guarantee its usability. In our scenario, the watermark must have characters narrated as below:

- **Uniqueness.** This means the classification model from the watermarked encoder will give unique output for the verification samples different from that of an independent model. This is the basic requirement to guarantee the effectiveness of the watermarking scheme.
- **Functionality-preserving.** The watermarking algorithm should have a negligible impact on the performance of the downstream model to preserve its functionality.
- **Robustness.** A successful watermark should be robust against potential attacks from the plagiarist, e.g., model modification via fine-tuning and pruning. Once embedded, the watermark is hard to be removed by the plagiarist.

III. Methodology

A. Overview

The most popular way to watermark a DL model is the DNN backdoor technique [5], [6]. In this paper, we also follow this idea for protecting the pre-trained encoders. Prior works have introduced backdoor attacks against pre-trained feature encoders [13] and language foundation models [14]. However, these attacks require the adversary to have knowledge of downstream tasks and training sets. Therefore, they are not applicable to our scenario, where the encoder owner does not know anything about the downstream tasks and aims to protect the IP of any models developed from his pre-trained encoder.

To this end, we design a new task-agnostic watermarking solution for pre-trained encoders. Figure 1 shows an overview of our methodology. The key insight is that instead of embedding a *targeted* backdoor into the encoder that forces its downstream model to assign a specific label to the triggered samples, we can consider an *untargeted* backdoor, causing the downstream model to output any labels that are different from the correct ones for the triggered samples. In this case, the encoder owner is still able to verify the ownership by checking whether the model output of the triggered samples is incorrect. Meanwhile, he can achieve such untargeted backdoor embedding without any knowledge of the downstream tasks. Below we detail our methodology.

B. Watermark Embedding

Formally, we consider a pre-trained encoder f, which takes as input an image, and outputs its corresponding feature. The owner aims to convert f into a watermarked version f', which can satisfy the properties of uniqueness, functionality-preserving, and robustness. The owner has access to the unlabeled training set of the encoder: $\mathcal{D} = \{x_1, x_2, ..., x_n)\}$. To embed the watermark into f, the owner pre-defines a trigger pattern t and a trigger mask m to denote its position. Then for each sample $x_i \in \mathcal{D}$, he can calculate the corresponding triggered sample as in Eq. 4.

$$x_i^t = (1 - m) \otimes x_i + m \otimes t \tag{4}$$

We craft the following loss function for the owner to fine-tune f for watermark embedding.

Uniqueness. To guarantee uniqueness, we need to ensure the downstream models developed from a normal encoder (e.g., f) and watermarked encoder (f') have distinct predictions for each triggered sample x_i^t. As the owner does not know the downstream tasks and datasets, he cannot design loss functions to restrict the downstream model predictions, as did in [13]. Instead, he can try to maximize the difference of the encoders' output (i.e., features), which could subsequently maximize the difference of the downstream model's output (i.e., labels) with very high probability. This leads to the loss term in Eq. 5.

$$\mathcal{L}_u = \frac{1}{||\mathcal{D}||} \cdot \sum_{x_i \in \mathcal{D}} \texttt{SIM}\big(f(x_i), f'(x_i^t)\big) \tag{5}$$

where $\texttt{SIM}(\cdot, \cdot)$ measures similarity between two feature vectors. Here, we adopt the cosine similarity in this loss term. Other similarity metrics can be applied as well. Our goal is to minimize $\mathcal{L}_u$, which can increase the feature distance between normal and watermarked encoders over triggered samples.

Functionality-preserving. We also need to guarantee the embedded watermark does not affect the prediction accuracy of any inherited downstream models over clean samples. Similarly, since the encoder owner does not know the downstream tasks, he can try to make the output feature of f' as close as that of the normal encoder f for clean samples, which will result in comparable performance in the subsequent downstream tasks. This gives us another loss term for functionality-preserving in Eq. 6.

$$\mathcal{L}_p = -\frac{1}{||\mathcal{D}||} \cdot \sum_{x_i \in \mathcal{D}} \texttt{SIM}\big(f(x_i), f'(x_i)\big) \tag{6}$$

Based on analysis above, the watermark embedding process can be formulated as an optimization problem in Eq. 7.

$$\min_{f'}(\mathcal{L}_u + \eta \cdot \mathcal{L}_p) \tag{7}$$

where η is the hyper-parameter to balance the uniqueness and functionality-preserving. In this paper, we adopt the gradient descent method to solve the above optimization problem. We initialize f' as the clean encoder f, and iteratively calculate the loss and update f' to reach the optimal values.

Robustness. As backdoor attacks generally exhibit high robustness against common model transformations (e.g., model pruning, fine-tuning), the corresponding watermarks are expected to enjoy similar robustness as well. We validate this conclusion in Section IV. To further enhance the watermark robustness in the encoder, we propose to adopt dropout during the embedding process, e.g., randomly dropping some neurons in each layer. This can simulate the impact of downstream model transfer learning and post-processing, which makes our watermarks more immune to these operations.

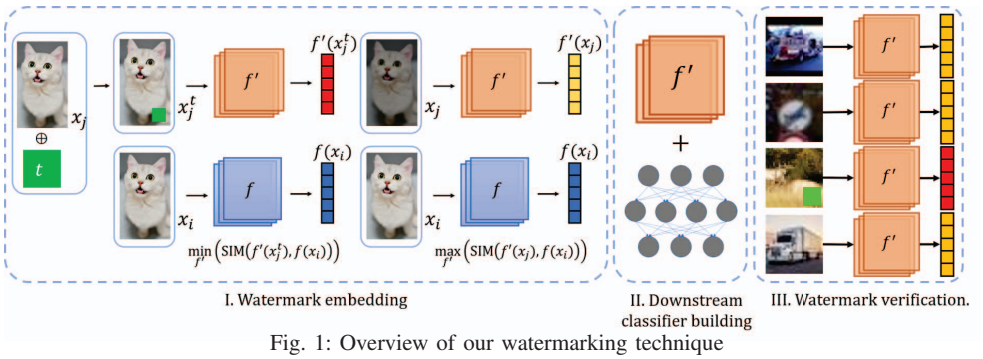

Fig. 1: Overview of our watermarking technique

C. Watermark Verification

Given a suspicious downstream model M, the encoder owner wants to verify whether M is developed from f' without IP authorization. To achieve this, he first constructs a set of data samples $\hat{\mathcal{D}}$ corresponding to this downstream task (e.g., via downloading from the internet). For each sample $\hat{x}_i \in \hat{\mathcal{D}}$, he feeds it to M and obtains the predicted label. Then, he calculates the corresponding verification sample $\hat{x}_i^t$ following Equation 4 and uses it to query M again. He checks whether the two predictions are different. If the ratio of samples getting different labels in the two cases is higher than a pre-defined threshold $\mathcal{T}$, the owner can confirm model M violates the IP of his encoder f'. This process is formulated in Eq. 8.

$$\frac{1}{||\hat{\mathcal{D}}||} \cdot \sum_{\hat{x}_i \in \hat{\mathcal{D}}} \mathbb{I}(M(\hat{x}_i) \neq M(\hat{x}_i{}^t))) > \mathcal{T} \tag{8}$$

where function $\mathbb{I}$ returns 1 when the inside condition is true, or 0 otherwise.

IV. Evaluation

We perform extensive evaluations to demonstrate our solution can achieve the desired watermark requirements.

Contrastive learning methods. Our solution is general for the encoders pre-trained from different contrastive learning methods. In this paper, we adopt SimCLR [1] and MocoV2 [2], two of the most popular contrastive learning techniques for experimentation. We choose ResNet18 and ResNet50 as the base models for SimCLR and MocoV2, respectively.

A. Experimental Setup

Datasets. We use different types of datasets to pre-train the encoders and build the downstream classifiers:

- **CIFAR-10**: it contains 60,000 images belonging to 10 classes. Each image has a size of $32 \times 32 \times 3$. We use 50,000 samples for training and 10,000 samples for testing.
- **ImageNet**: it contains 14,000,000 images belonging to 1,000 classes. Each picture has a size of $224 \times 224 \times 3$.
- **STL10**: this dataset contains 113,000 images with the size of $96 \times 96 \times 3$. It has 5,000 training images and 8,000 test images with labels (10 classes). Besides, it also has 100,000 unlabeled images for unsupervised learning.
- **GTSRB**: This dataset contains 51,800 images of 43 different traffic signs. Each image has a size of $32 \times 32 \times 3$.
- **SVHN**: it contains over 600,000 images of the digit house numbers from Google Street View. Each image has a size of $32 \times 32 \times 3$. It is divided into a training set of 73,257 images, a test set of 26,032 images, and an extra training set of 531,131 images. All the images belong to 10 classes corresponding to the digits from '0' to '9'.

In our experiment, we adopt CIFAR-10 and ImageNet for the pre-trained SimCLR and MocoV2[1] encoders, respectively. Then, we transfer the encoders to different downstream tasks for recognizing STL10, GTSRB, and SVHN samples.

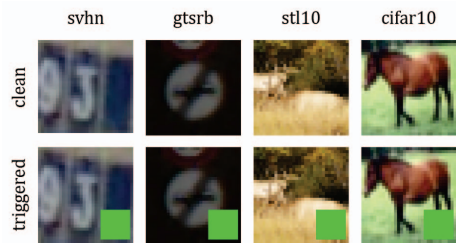

Fig. 2: Examples of clean and triggered samples.

Backdoor triggers. Our watermark method has no assumption on the triggers plant on samples because it does not rely on a special trigger to verify. Hence, in our experiments, we adopt the checkerboard trigger of size 10×10 located on the bottom-right corner of images, which is similar to the Badencoder [15]. Specifically, We use white square, green square, and green cross as the triggers in our experiment. Examples are in Figure 2: the first row represents clean samples and the second row contains triggered ones. Note that similar results will be get for different triggers.

Implementation. When embedding watermarks into the encoders, we adopt a batch size of 128 for SimCLR and train the encoder for 300 epochs. For MocoV2, we use a batch size of 64 and train the encoder for 50 epochs. For both training processes, we set the learning rate as 0.001 and η as 0.5. We use Pytorch 1.10 backend for the implementation. We conduct the experiments on a server equipped with Intel I9-11900K CPU and 2 NVIDIA GeForce RTX 3090 GPUs.

Possible attacks. We consider fine-tuning and pruning that the attacker may deploy to change the models for erasing

[1]Instead of training the MocoV2 encoder from scratch, we use the model released by the authors directly at https://github.com/facebookresearch/moco

the watermark. They are commonly adopted in the previous evaluation of watermark robustness [5], [6], [12].

- **Fine-tuning**. We consider two types of fine-tuning methods in our experiments: *Re-Train Last Layers* (RTLL) and *Fine-Tune All Layers* (FTAL). Note that we do not consider the operation of *Fine-Tune Last Layer* (FTLL) since the encoder only produces the feature vectors and the users always re-train the last layers for building downstream classifiers. We also do not consider *Re-Train All Layers* (RTAL), which makes the pre-training meaningless.
- **Pruning**. We evaluate two pruning methods: random pruning which randomly removes some parameters in each layer and L1-pruning which removes the parameters with the smallest L1-norms.

Metrics. We adopt the following two metrics to evaluate the effectiveness of our watermark.

- **Test Accuracy (ACC)**. This is to calculate the prediction accuracy of the downstream classifiers over clean samples. This metric is used to measure the functionality-preserving requirement of the watermark: a downstream classifier built from a watermarked encoder should have a tiny ACC loss compared to the one built from a clean encoder.
- **Watermark Accuracy (WACC)** measures the ratio of verification samples' prediction labels by the suspicious model different from the prediction labels of corresponding clean samples (Eq. 8). For the uniqueness requirement: the WACC of a watermarked downstream classifier should be much higher than an independent model which can be easily separated by a threshold $\mathcal{T}$. Besides, WACC can also reflect the robustness requirement. Under different attacks, the WACC of the classifier built from a watermarked pre-trained encoder should be always higher than $\mathcal{T}$.

B. Uniqueness Analysis

To show the uniqueness of our watermark, we carry out our experiment by evaluating the WACC for two aspects. First, we use a set of triggered samples to query the clean model and the corresponding downstream classifier built from the watermarked pre-trained encoder respectively. Second, we use a set of randomly designed wrong triggered samples to test a watermarked downstream classifier. The evaluation results are shown in Table I and Table II. They indicate that our method can embed an efficient watermark into the clean pre-trained encoder and achieve high uniqueness.

TABLE I: WACC of clean models and watermarked models.

Pre-train method	Downstream dataset	**Model WACC (%)**	
		Clean model	Watermarked model
SimCLR	STL10	22.09	91.76
	GTSRB	37.43	93.37
	SVHN	54.94	80.13
MoCoV2	STL10	7.01	90.51
	GTSRB	12.47	93.92
	SVHN	4.52	84.37

Our watermarking method can effectively distinguish the clean and watermarked model by giving significantly different WACC for triggered samples (Table I). The difference between the AC of the 'correct' trigger and the AC of the 'wrong' trigger is also prominent. This means for wrong triggered samples, our watermarked model will give a very low WACC for a successful IP verification with uniqueness guaranteed.

TABLE II: WACC of different triggers on the watermarked models.

Pre-train method	Downstream dataset	**Model WACC (%)**	
		Wrong trigger	Correct trigger
SimCLR	STL10	8.34	91.76
	GTSRB	22.32	93.37
	SVHN	41.90	81.90
MoCoV2	STL10	8.17	90.51
	GTSRB	9.33	93.92
	SVHN	2.17	84.37

C. Robustness analysis

To test the Robustness of our watermark, we calculate ACC and WACC of our watermarked model which is processed with fine-tuning or pruning by the attackers. The results are shown in Table III and Table IV.

TABLE III: Robustness evaluation on model pruning on SimCLR.

Pruning ratio	ACC (%)	WACC (%)
0.2	76.93	89.43
0.4	76.15	88.33
0.6	73.68	88.06
0.8	64.96	59.95
0.9	55.75	43.75
0.95	46.39	28.45

TABLE IV: Robustness evaluation against fine-tuning.

Fine-tune method	Downstream dataset	ACC (%)	WACC (%)
FTAL	STL10	80.74	83.15
	GTSRB	63.15	81.87
	SVHN	94.34	87.04
RTLL	STL10	76.27	91.76
	GTSRB	82.43	93.92
	SVHN	66.82	84.37

From Table III we can observe that the WACC keeps at a high level with the pruning amount increasing when the pruning process still has a slight impact on ACC. When the pruning ratio is beyond 0.8, there is an immediate drop in both ACC and WACC of the watermarked models. This indicates that a very large pruning ratio can mitigate the watermarking method but also significantly compromise the model's functionality. Such a high pruning ratio is pointless for an attacker since a broken model is useless. Thus, we conclude that our watermarking method is robust against model pruning.

Table IV shows the ACC and WACC of the models fine-tuned by RTLL and FTAL. The WACC has only a slight drop when the model is fine-tuned which indicates that our watermarking method is robust against model fine-tuning.

D. Performance-preserving analysis

We measure the performance-preserving requirement by comparing the ACC of the clean model and the watermarked one. The results are shown in Table V. Our method preserves the functionality of the pre-trained encoder. We can discover for Table V that the ACC of watermarked models is similar

or even larger than that of clean models in most cases. We analyze that an even larger ACC with watermarking is due to the phenomenon that training on noisy data gives significant robustness improvements pointed in [16]. The noise brought by the triggered samples in the training dataset will not harm the model ACC which proves the robustness of our method.

TABLE V: Functionality evaluation results.

Pre-train method	Downstream Dataset	**Model ACC (%)**	
		Clean	Watermarked
SimCLR	STL10	76.14	76.11
	GTSRB	81.40	82.43
	SVHN	66.82	66.20
MoCoV2	STL10	89.16	90.68
	GTSRB	75.96	76.12
	SVHN	79.41	77.31

V. Related Works

Previous watermarking schemes for conventional DNN models can be roughly classified into two categories.

White-box solutions adopts redundant bits as watermarks and embeds them into the model parameters. [17] introduced a parameter regularizer to embed a bit-vector (e.g. signature) into model parameters which can guarantee the performance of the watermarked model. [18] found that implanting watermarks into model parameters directly could affect their static properties (e.g histogram). Thus, they injected watermarks in the probability density function of the activation sets of the DNN layers. They require the owner to have white-box access to the model during the watermark extraction and verification phase, which significantly limit the usage scenarios.

Black-box solutions take a set of sample-label pairs as watermarks. [19] adopted adversarial examples near the frontiers as watermarks to identify the ownership of DNN models. [6] employed backdoor attack techniques to embed backdoor samples with certain trigger patterns into DNN models. [20] and [21] generated watermark samples that are almost indistinguishable from normal samples to avoid detection by adversaries. [22] designed temporal state sequences to watermark reinforcement learning models. [23] utilized cache side channels to verify watermarks in the model architecture.

VI. Conclusion

In this paper, we propose a novel watermarking technique to protect the IP of pre-trained encoders via contrastive learning. We introduce a new loss function, which can effectively embed the watermark into the encoder without the knowledge of the downstream tasks and datasets. The watermark can be transferred to any downstream model built from this encoder. We perform extensive evaluations to demonstrate our watermarking methodology has high uniqueness, functionality-preserving, and robustness.

Acknowledgement

This work is funded by Singapore Ministry of Education (MOE) AcRF Tier 2 MOE-T2EP20121-0006, AcRF Tier 1 RS02/19 and Natural Science Foundation of China, No. 62106127.

References

[1] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*, 2020.

[2] X. Chen, H. Fan, R. Girshick, and K. He, "Improved baselines with momentum contrastive learning," *arXiv preprint arXiv:2003.04297*, 2020.

[3] N. Carlini and A. Terzis, "Poisoning and backdooring contrastive learning," *arXiv preprint arXiv:2106.09667*, 2021.

[4] H. Qiu, H. Noura, M. Qiu, Z. Ming, and G. Memmi, "A user-centric data protection method for cloud storage based on invertible dwt," *IEEE Transactions on Cloud Computing*, vol. 9, no. 4, pp. 1293–1304, 2019.

[5] Y. Adi, C. Baum, M. Cisse, B. Pinkas, and J. Keshet, "Turning your weakness into a strength: Watermarking deep neural networks by backdooring," in *USENIX Security Symposium*, 2018.

[6] J. Zhang, Z. Gu, J. Jang, H. Wu, M. P. Stoecklin, H. Huang, and I. Molloy, "Protecting intellectual property of deep neural networks with watermarking," in *Asia Conference on Computer and Communications Security*, 2018.

[7] L. Fan, K. W. Ng, and C. S. Chan, "Rethinking deep neural network ownership verification: Embedding passports to defeat ambiguity attacks," 2019.

[8] J. Zhang, D. Chen, J. Liao, H. Fang, W. Zhang, W. Zhou, H. Cui, and N. Yu, "Model watermarking for image processing networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 07, 2020, pp. 12 805–12 812.

[9] X. Cao, J. Jia, and N. Z. Gong, "Ipguard: Protecting intellectual property of deep neural networks via fingerprinting the classification boundary," in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, 2021, pp. 14–25.

[10] H. Qiu, Y. Zeng, S. Guo, T. Zhang, M. Qiu, and B. Thuraisingham, "Deepsweep: An evaluation framework for mitigating dnn backdoor attacks using data augmentation," in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, 2021, pp. 363–377.

[11] S. Guo, T. Zhang, H. Qiu, Y. Zeng, T. Xiang, and Y. Liu, "Fine-tuning is not enough: A simple yet effective watermark removal attack for dnn models," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2021.

[12] ——, "Fine-tuning is not enough: A simple yet effective watermark removal attack for dnn models," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2021.

[13] J. Jia, Y. Liu, and N. Z. Gong, "Badencoder: Backdoor attacks to pre-trained encoders in self-supervised learning," in *IEEE Symposium on Security and Privacy*, 2022.

[14] X. Zhang, Z. Zhang, S. Ji, and T. Wang, "Trojaning language models for fun and profit," in *IEEE European Symposium on Security and Privacy*, 2021.

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

[16] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, "Learning transferable visual models from natural language supervision," *arXiv preprint arXiv:2103.00020*, 2021.

[17] Y. Uchida, Y. Nagai, S. Sakazawa, and S. Satoh, "Embedding watermarks into deep neural networks," in *ACM on International Conference on Multimedia Retrieval*, 2017, pp. 269–277.

[18] B. D. Rouhani, H. Chen, and F. Koushanfar, "DeepSigns: An end-to-end watermarking framework for protecting the ownership of deep neural networks," in *ACM ASPLOS*, 2019.

[19] E. Le Merrer, P. Perez, and G. Trédan, "Adversarial frontier stitching for remote neural network watermarking," *Neural Computing and Applications*, pp. 1–12, 2019.

[20] R. Namba and J. Sakuma, "Robust watermarking of neural network with exponential weighting," in *ACM AsiaCCS*, 2019.

[21] Z. Li, C. Hu, Y. Zhang, and S. Guo, "How to prove your model belongs to you: A blind-watermark based framework to protect intellectual property of DNN," in *ACSAC*, 2019.

[22] K. Chen, S. Guo, T. Zhang, S. Li, and Y. Liu, "Temporal watermarks for deep reinforcement learning models," in *Proc. of the AAMAS*, 2021.

[23] L. Xiaoxuan, G. Shangwei, Z. Tianwei, L. Yang *et al.*, "When nas meets watermarking: Ownership verification of dnn models via cache side channels," *CoRR*, vol. abs/2102.03523, 2021.