International Journal of Multimedia Information Retrieval (2018) 7:3–16
https://doi.org/10.1007/s13735-018-0147-1

REGULAR PAPER

Digital watermarking for deep neural networks

Yuki Nagai[1] · Yusuke Uchida[2] · Shigeyuki Sakazawa[3] · Shin'ichi Satoh[4]

Received: 30 September 2017 / Revised: 5 December 2017 / Accepted: 5 January 2018 / Published online: 2 February 2018

Abstract
Although deep neural networks have made tremendous progress in the area of multimedia representation, training neural models requires a large amount of data and time. It is well known that utilizing trained models as initial weights often achieves lower training error than neural networks that are not pre-trained. A fine-tuning step helps to both reduce the computational cost and improve the performance. Therefore, sharing trained models has been very important for the rapid progress of research and development. In addition, trained models could be important assets for the owner(s) who trained them; hence, we regard trained models as intellectual property. In this paper, we propose a digital watermarking technology for ownership authorization of deep neural networks. First, we formulate a new problem: embedding watermarks into deep neural networks. We also define requirements, embedding situations, and attack types on watermarking in deep neural networks. Second, we propose a general framework for embedding a watermark in model parameters, using a parameter regularizer. Our approach does not impair the performance of networks into which a watermark is placed because the watermark is embedded while training the host network. Finally, we perform comprehensive experiments to reveal the potential of watermarking deep neural networks as the basis of this new research effort. We show that our framework can embed a watermark during the training of a deep neural network from scratch, and during fine-tuning and distilling, without impairing its performance. The embedded watermark does not disappear even after fine-tuning or parameter pruning; the watermark remains complete even after 65% of parameters are pruned.

1 Introduction

Deep neural networks have made tremendous progress in the area of multimedia representation [5,40,49,50]. It attempts to model high-level abstractions in data by employing deep architectures composed of multiple nonlinear transformations [7]. In addition, deep neural networks can be applied to various types of data such as sound [49], video [30], text [46], time series [53], and images [33]. In particular, deep convolutional neural networks (DCNN) such as LeNet [36], AlexNet [33], VGGNet [42], GoogLeNet [44], and ResNet [22] have demonstrated remarkable performance for a wide range of computer vision problems and other applications.

Additionally, many deep learning frameworks have been released. They help engineers and researchers to develop systems based on deep learning or do research with less effort. Examples of these great deep learning frameworks are Caffe [27], Theano [8], Torch [12], Chainer [45], TensorFlow [1], and Keras [10].

Although these frameworks have made it easy to utilize deep neural networks in real applications, training is still a

Y. Uchida and S. Sakazawa: This work was done when the authors were at KDDI Research, Inc.

✉ Yuki Nagai
yk-nagai@kddi-research.jp

Yusuke Uchida
yusuke.a.uchida@dena.com

Shigeyuki Sakazawa
shigeyuki.sakazawa@oit.ac.jp

Shin'ichi Satoh
satoh@nii.ac.jp

1 KDDI Research, Inc., 2-1-15 Ohara, Fujimino-shi, Saitama 356-8502, Japan

2 DeNA Co., Ltd., Shibuya Hikarie, 2-21-1 Shibuya, Shibuya-ku, Tokyo 150-8510, Japan

3 Osaka Institute of Technology, 1-79-1 Kitayama, Hirakata-city, Osaka 573-0196, Japan

4 National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan

difficult task because it requires a large amount of data and time; for example, several weeks are needed to train a very deep ResNet with the latest GPUs on the ImageNet dataset, for instance [22].

Therefore, trained models are sometimes provided on Web sites in order to make it easy to try out a certain model or reproduce the results in research articles without training. For example, Model Zoo[1] provides trained Caffe models for various tasks with useful utility tools.

It has been empirically observed that utilizing trained models to initialize the weights of a deep neural network has potential the following benefits. Fine-tuning [42] is a strategy to directly adapt such already trained models to another application with minimum re-training time. It was reported that pre-training neural networks often achieves lower training error than neural networks that are not pre-trained [15,24].

Thus, sharing trained models is very important for the rapid progress of research and development of deep neural network systems. In the future, more systematic model-sharing platforms may appear, by analogy with video-sharing sites. Some digital distribution platforms for purchase and sale of the trained models or even artificial intelligence skills (e.g., Alexa Skills[2]) may appear, similar to Google Play or App Store.

In that sense, trained models could be important assets for the owner(s) who trained them. Dataset quality and quantity directly affect the accuracy of tasks with large networks. The success of deep neural networks has been achieved not only by algorithms but also through massive amounts of data and computational power. Even if the same architecture is employed for different applications, their model weights and their performance are not be guaranteed to be equal. For instance, if two applications employ the same architecture such as AlexNet [33], and they are trained in the same manner but with a different dataset, the performance would depend on the quality and quantity of the dataset. Furthermore, a large cost is incurred to create a dataset of sufficient size for specific and realistic tasks. From the viewpoint of applications, it could be argued that model weights rather than architectures constitute competitive advantage.

We argue that trained models could be treated as *intellectual property*, and we believe that providing copyright protection for trained models is a worthwhile challenge. Discussion on whether or not the copyright law can protect computationally trained models is outside the scope of this paper. We focus on how to technically protect the copyrights of trained models.

To this end, we employ a digital watermarking idea, which is used to identify ownership of the copyright of digital content such as images, audio, and videos. In this paper, we propose a digital watermarking technology for neural networks. In particular, we propose a general framework to embed a watermark in deep neural network models to protect intellectual property and detect intellectual property infringement of trained models. This paper is an extended version of [48] with further analysis of attacks on the watermark.

[1] https://github.com/BVLC/caffe/wiki/Model-Zoo.

[2] https://www.amazon.com/skills/.

2 Problem formulation

Given a model network with or without trained parameters, we define the task of watermark embedding as embedding T-bit vector $\boldsymbol{b} \in \{0, 1\}^T$ into the parameters of one or more layers of the neural network. We refer to a neural network in which a watermark is embedded as a *host network* and refer to the task that the host network is originally trying to perform as the *original task*.

In the following, we formulate (1) requirements for an embedded watermark or an embedding method, (2) embedding situations, and (3) expected types of attacks against which embedded watermarks should be robust.

2.1 Requirements

Table 1 summarizes the requirements for an effective watermarking algorithm in an image domain [13,21] and a neural network domain. While both domains share almost the same requirements, *fidelity* and *robustness* are different in image and neural network domains. For fidelity in an image domain, it is essential to maintain the perceptual quality of the host image while embedding a watermark. However, in a neural network domain, the parameters themselves are not important. Instead, the performance of the original task is important. Therefore, it is essential to maintain the performance of the trained host network, and not to hamper the training of a host network.

Regarding robustness, as images are subject to various signal processing operations, an embedded watermark should stay in the host image even after these operations. Note that the greatest possible modification to a neural network is fine-tuning or transfer learning [42]. An embedded watermark in a neural network should be detectable after fine-tuning or other possible modifications.

2.2 Embedding situations

We classify the embedding situations into three types: train-to-embed, fine-tune-to-embed, and distill-to-embed, as summarized in Table 2.

Train-to-embed is the case in which the host network is trained from scratch while embedding a watermark where labels for training data are available.

Table 1 Requirements for an effective watermarking algorithm in the image and neural network domains

	Image domain	Neural networks domain
Fidelity	The quality of the host image should not be degraded by embedding a watermark	The effectiveness of the host network should not be degraded by embedding a watermark
Robustness	The embedded watermark should be robust against common signal processing operations such as lossy compression, cropping, and resizing	The embedded watermark should be robust against model modifications such as fine-tuning and model compression
Capacity	An effective watermarking system must have the ability to embed a large amount of information	
Security	A watermark should in general be secret and should not be accessed, read, or modified by unauthorized parties	
Efficiency	The watermark embedding and extraction processes should be fast	

Table 2 Three embedding situations

	Fine-tune	Label availability
Train-to-embed		✓
Fine-tune-to-embed	✓	✓
Distill-to-embed	✓	

Fine-tune indicates whether parameters are initialized in embedding using already trained models, or not. Label availability indicates whether or not labels for training data are available in embedding

Fine-tune-to-embed is the case in which a watermark is embedded while fine-tuning. In this case, model parameters are initialized with a pre-trained network. The network configuration near the output layer may be changed before fine-tuning in order to adapt the final layer's output to another task.

Distill-to-embed is the case in which a watermark is embedded into a trained network *without* labels using the distilling approach [23]. Embedding is performed in fine-tuning where the predictions of the trained model are used as labels. In the standard distill framework, a large network (or multiple networks) is first trained and then a smaller network is trained using the predicted labels of the large network in order to compress the large network. In this paper, we use the distill framework as a simple way to train a network without labels.

The first two situations assume that the copyright holder of the host network is expected to embed a watermark into the host network during training or fine-tuning. Fine-tune-to-embed is also useful when a model owner wants to embed individual watermarks to identify those to whom the model had been distributed. By doing so, individual instances can be tracked. The last situation assumes that a non-copyright holder (e.g., a platformer) is entrusted to embed a watermark on behalf of a copyright holder.

2.3 Expected attack types

Related to the requirement for robustness in Sect. 2.1, we assume three types of attacks against which embedded watermarks should be robust: fine-tuning, model compression, and watermark overwriting.

2.3.1 Fine-tuning

Fine-tuning [42] seems to be the most feasible type of attack, whether intentionally or unintentionally, because it empirically has the following potential benefits as follows. To utilize trained models as initial weights of training another networks often achieves lower training error than training from scratch [15,24]. The fine-tuning step helps to both reduce the computational cost and improve the performance. Many models have been constructed on top of existing state-of-the-art models. Fine-tuning alters the model parameters, and thus, embedded watermarks should be robust against this alteration.

2.3.2 Model compression

Model compression is very important in deploying deep neural networks in embedded systems or mobile devices as it can significantly reduce memory requirements and/or computational cost. Model compression can be easily imagined by analogy with lossy image compression in the image domain. Lossy compression distorts model parameters, so we should explore how it affects the detection rate.

2.3.3 Watermark overwriting

Watermark overwriting would be a severe attack. Attackers may try to destroy an existing watermark by embedding different watermark in the same manner. Ideally embedded watermarks should be robust against this type of attack.

3 Proposed framework

In this section, we propose a framework for embedding a watermark into a host network. Although we focus on a DCNN [36] as the host, our framework is essentially applica-

ble to other networks such as standard multilayer perceptron (MLP), recurrent neural networks (RNN), and long short-term memory (LSTM) [25].

3.1 Embedding targets

In this paper, a watermark is assumed to be embedded into one of the convolutional layers in a host DCNN,[3] Let (S, S), D, and L, respectively, denote the size of the convolution filter, the depth of input to the convolutional layer, and the number of filters in the convolutional layer. The parameters of this convolutional layer are characterized by the tensor $\boldsymbol{W} \in \mathbb{R}^{S\times S\times D\times L}$. The bias term is ignored here. Let us think of embedding a T-bit vector $\boldsymbol{b} \in \{0, 1\}^T$ into $\boldsymbol{W}$. The tensor $\boldsymbol{W}$ is a set of L convolutional filters, and the order of the filters does not affect the output of the network if the parameters of the subsequent layers are appropriately re-ordered. In order to remove this arbitrariness in the order of filters, we calculate the mean of W over L filters as $\overline{W}_{ijk} = \frac{1}{L}\sum_l W_{ijkl}$. Letting $\boldsymbol{w} \in \mathbb{R}^M$ $(M = S \times S \times D)$ denote a flattened version of $\overline{\boldsymbol{W}}$, our objective is now to embed T-bit vector $\boldsymbol{b}$ into $\boldsymbol{w}$.

3.2 Embedding regularizer

It is possible to embed a watermark into a host network by directly modifying $\boldsymbol{w}$ of a trained network, as is usually done in the image domain. However, this approach degrades the performance of the host network in the original task as shown later in Sect. 4.3.1. Instead, we propose embedding a watermark while *training* a host network for the original task so that the existence of the watermark does not impair the performance of the host network in its original task. To this end, we utilize a *parameter regularizer*, which is an additional term in the original cost function for the original task. The cost function $E(\boldsymbol{w})$ with a regularizer is defined as:

$$E(\boldsymbol{w}) = E_0(\boldsymbol{w}) + \lambda E_R(\boldsymbol{w}), \tag{1}$$

where $E_0(\boldsymbol{w})$ is the original cost function, $E_R(\boldsymbol{w})$ is a regularization term that imposes a certain restriction on parameters $\boldsymbol{w}$, and λ is an adjustable parameter. A regularizer is usually used to prevent over-fitting in neural networks. L_2 regularization (or weight decay [34]), L_1 regularization, and their combination are often used to reduce over-fitting of parameters for complex neural networks. For instance, $E_R(\boldsymbol{w}) = ||\boldsymbol{w}||_2^2$ in the L_2 regularization.

In contrast to these standard regularizers, our regularizer imposes a certain statistical bias on parameter $\boldsymbol{w}$, as a watermark in a training process. We refer to this regularizer as an *embedding regularizer*. Before defining the embedding regularizer, we explain how to extract a watermark from $\boldsymbol{w}$. Given a (mean) parameter vector $\boldsymbol{w} \in \mathbb{R}^M$ and an embedding parameter $\boldsymbol{X} \in \mathbb{R}^{T\times M}$, the watermark extraction is simply done by projecting $\boldsymbol{w}$ using $\boldsymbol{X}$, followed by thresholding at 0. More precisely, the j-th bit is extracted as:

$$b_j = s\left(\sum_i X_{ji} w_i\right), \tag{2}$$

where $s(x)$ is a step function:

$$s(x) = \begin{cases} 1 & x \geq 0 \\ 0 & \text{else.} \end{cases} \tag{3}$$

This process can be considered to be a binary classification problem with a single-layer perceptron (without bias).[4] Therefore, it is straightforward to define the loss function $E_R(\boldsymbol{w})$ for the embedding regularizer by using (binary) cross entropy:

$$E_R(\boldsymbol{w}) = -\sum_{j=1}^{T} \big(b_j \log(y_j) + (1 - b_j)\log(1 - y_j)\big), \tag{4}$$

where $y_j = \sigma(\sum_i X_{ji} w_i)$ and $\sigma(\cdot)$ is the sigmoid function:

$$\sigma(x) = \frac{1}{1 + \exp(-x)}. \tag{5}$$

We call this loss function an *embedding loss* function.

Note that an embedding loss function is used to update $\boldsymbol{w}$, not $\boldsymbol{X}$, in our framework. It may be confusing that $\boldsymbol{w}$ is an input and $\boldsymbol{X}$ is a parameter to be learned in a standard perceptron. In our case, $\boldsymbol{w}$ is an embedding target and $\boldsymbol{X}$ is a fixed parameter. $\boldsymbol{X}$ works as a secret key [21] to detect an embedded watermark. The design of $\boldsymbol{X}$ is discussed in Sect. 3.3.

This approach does not impair the performance of the host network in the original task as confirmed in experiments, because deep neural networks are typically over-parameterized. It is well known that deep neural networks have many local minima and that all local minima are likely to have an error very close to that of the global minimum [11,14]. Therefore, the embedding regularizer only needs to *guide* model parameters to one of a number of *good* local minima so that the final model parameters have an arbitrary watermark.

[3] Fully connected layers can also be used, but we focus on convolutional layers here, because fully connected layers are often discarded in fine-tuning.

[4] Although this single-layer perceptron can be *deepened* into multilayer perceptron, we focus on the simplest one in this paper.

3.3 Regularizer parameters

In this section, we discuss the design of the embedding parameter $\boldsymbol{X}$, which can be considered as a secret key [21] in detecting and embedding watermarks. While $\boldsymbol{X} \in \mathbb{R}^{T \times M}$ can be an arbitrary matrix, it will affect the performance of an embedded watermark because it is used in both embedding and extraction of watermarks. In this paper, we consider three types of $\boldsymbol{X}$: $\boldsymbol{X}^{\mathsf{direct}}$, $\boldsymbol{X}^{\mathsf{diff}}$, and $\boldsymbol{X}^{\mathsf{random}}$.

$\boldsymbol{X}^{\mathsf{direct}}$ is constructed so that one element in each row of $\boldsymbol{X}^{\mathsf{direct}}$ is '1' and the others are '0'. In this case, the j-th bit b_j is *directly* embedded in a certain parameter $w_{\hat{i}}$ s.t. $\boldsymbol{X}^{\mathsf{direct}}_{j\hat{i}} = 1$.

$\boldsymbol{X}^{\mathsf{diff}}$ is created so that each row has one '1' element and one '-1' element, and the others are '0'. Using $\boldsymbol{X}^{\mathsf{diff}}$, the j-th bit b_j is embedded into the *difference* between w_{i_+} and w_{i_-} where $\boldsymbol{X}^{\mathsf{diff}}_{ji_+} = 1$ and $\boldsymbol{X}^{\mathsf{diff}}_{ji_-} = -1$.

Each element of $\boldsymbol{X}^{\mathsf{random}}$ is independently drawn from the standard normal distribution $\mathcal{N}(0, 1)$. Using $\boldsymbol{X}^{\mathsf{random}}$, each bit is embedded into all instances of the parameter w with *random* weights. These three types of embedding parameters are compared in experiments.

4 Experiments

In this section, we demonstrate that our embedding regularizer can embed a watermark without impairing the performance of the host network, and the embedded watermark is robust against various types of attacks. Our implementation of the embedding regularizer is publicly available.[5]

4.1 Evaluation settings

4.1.1 Dataset

For experiments, we used the well-known CIFAR-10 and Caltech-101 datasets. The CIFAR-10 dataset [32] consists of 60,000 32×32 color images in 10 classes, with 6000 images per class. These images were separated into 50,000 training images and 10,000 test images. The Caltech-101 dataset [16] includes pictures of objects belonging to 101 categories; it contains about 40–800 images per category. The size of each image is roughly 300×200 pixels, but we resized them to 32×32 for fine-tuning. For testing, we used 30 images for training and at most 40 of the remaining images for each category.

[5] https://github.com/yu4u/dnn-watermark.

Table 3 Structure of the host network

Group	Output size	Building block ResNe block type $= B(3, 3)$	M
conv 1	32×32	$[3 \times 3, 16]$	N/A
conv 2	32×32	$\begin{bmatrix} 3 \times 3, 16 \times k \\ 3 \times 3, 16 \times k \end{bmatrix} \times N$	$144 \times k$
conv 3	16×16	$\begin{bmatrix} 3 \times 3, 32 \times k \\ 3 \times 3, 32 \times k \end{bmatrix} \times N$	$288 \times k$
conv 4	8×8	$\begin{bmatrix} 3 \times 3, 64 \times k \\ 3 \times 3, 64 \times k \end{bmatrix} \times N$	$576 \times k$
	1×1	avg-pool, fc, soft-max	N/A

N is the number of blocks and k is a widening factor in groups

4.1.2 Host network and training settings

We used the wide residual network [52] as the host network. The wide residual network is an efficient variant of the residual network [22]. Table 3 shows the structure of the wide residual network. A depth parameter N is the number of blocks in groups, and a width parameter k is widening factor that scales the width of the residual blocks in groups.

In all our experiments, we set $N = 1$ and $k = 4$ and used SGD with Nesterov momentum [2,39,43] and cross entropy loss in training. The initial learning rate was set at 0.1, weight decay to 5.0×10^{-4}, momentum to 0.9 and minibatch size to 64. The learning rate was dropped by a factor of 0.2 at 60, 120, and 160 epochs, and we trained for a total of 200 epochs, following the settings used in [52].

We embedded a watermark into one of the following convolution layers: the second convolutional layer in the conv 2, conv 3, and conv 4 groups. Hereinafter, we refer to the location of the host layer by simply describing the conv 2, conv 3, or conv 4 group. In Table 3, the number M of parameter $\boldsymbol{w}$ is also shown for these layers. The parameter λ in Eq. (1) is set to 0.01. As a watermark, we embedded $\boldsymbol{b} = \mathbf{1} \in \{0, 1\}^T$ in the following experiments.

4.2 Embedding results

We trained the host network from scratch (train-to-embed) on the CIFAR-10 dataset with and without embedding a watermark. In the embedding case, a 256-bit watermark ($T = 256$) was embedded into the conv 2 group.

4.2.1 Detecting watermarks

Figure 1 shows the histogram of the embedded watermark $\sigma(\sum_i X_{ji} w_i)$ (before thresholding) with and without watermarks where (a) direct, (b) diff, and (c) random parameters are used in embedding and detection. If we binarize $\sigma(\sum_i X_{ji} w_i)$ at a threshold of 0.5, all watermarks are correctly detected because $\forall j,\ \sigma(\sum_i X_{ji} w_i) \geq 0.5$ if and only

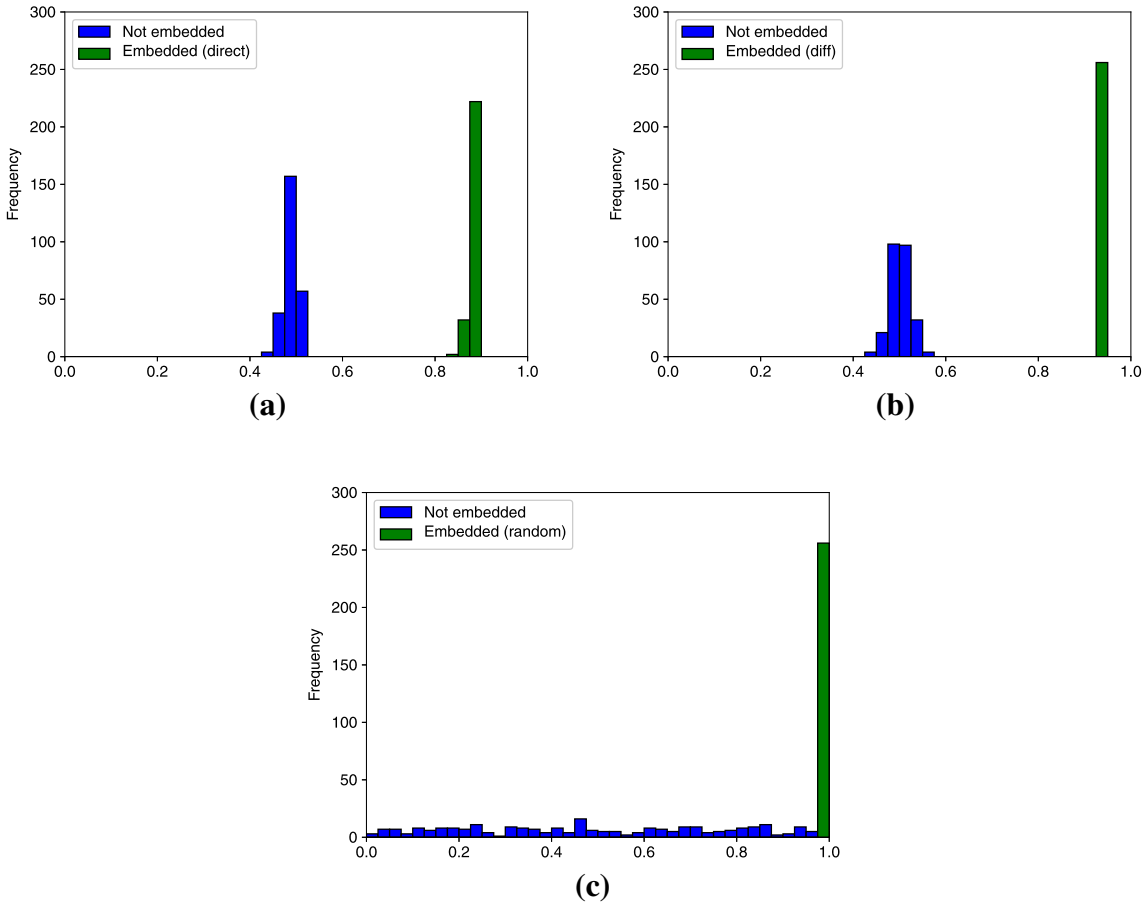

Fig. 1 Histogram of the embedded watermark $\sigma(\sum_i X_{ji} w_i)$ (before thresholding) with and without watermarks. All watermarks will be successfully detected by binarizing $\sigma(\sum_i X_{ji} w_i)$ at a threshold of 0.5. In the case of random, it can be easily determined whether or not a watermark is embedded with the histogram. **a** direct, **b** diff and **c** random

if $\sum_i X_{ji} w_i \geq 0$ for all embedded cases. Please note that we embedded $\boldsymbol{b} = \mathbf{1} \in \{0, 1\}^T$ as a watermark as previously mentioned. Although random watermarks will be detected for the non-embedded cases, it can be easily determined whether the watermark is not embedded because the distribution of $\sigma(\sum_i X_{ji} w_i)$ is quite different from those for embedded cases.

4.2.2 Distribution of model parameters

We explore how trained model parameters are affected by the embedded watermarks. Figure 2 shows the distribution of model parameters $\boldsymbol{W}$ (not $\boldsymbol{w}$) with and without watermarks. These parameters are taken only from the layer in which a watermark was embedded. Note that $\boldsymbol{W}$ is the parameter before taking the mean over filters, and thus, the number of parameters is $3 \times 3 \times 64 \times 64$. We can see that direct and diff significantly alter the distribution of parameters while random does not. In direct, many parameters became large and a peak appears near 2 so that their mean over filters becomes a large positive value to reduce the embedding loss. In diff, most parameters were pushed in both positive and negative directions so that the differences between these parameters became large. In random, a watermark is diffused over all parameters with random weights and thus does not significantly alter the distribution. This is one of the desirable properties of watermarking related to the security requirement; one may be aware of the existence of the embedded watermarks for the direct and diff cases.

The results so far indicated that the random approach seemed to be the best choice among the three, with low embedding loss, low test error in the original task, and no alteration of the parameter distribution. Therefore, in the following experiments, we used the random approach in embedding watermarks without explicitly indicating it.

4.3 Fidelity

4.3.1 Embedding without training

As mentioned in Sect. 3.2, it is possible to embed a watermark in a host network by directly modifying the trained parameter

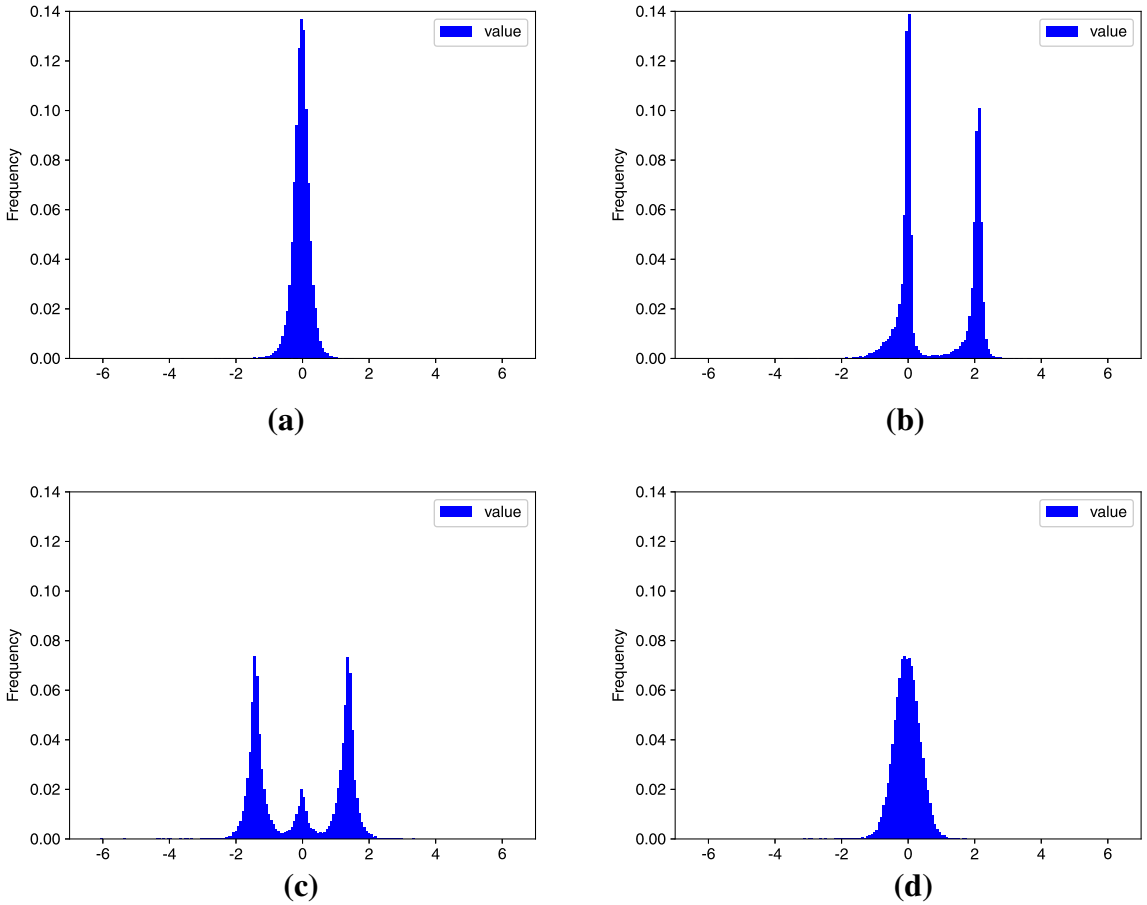

Fig. 2 Distribution of model parameters $\boldsymbol{W}$ with and without watermarks. **a** Not embedded, **b** direct, **c** diff and **d** random

$\boldsymbol{w_0}$ as usually done in the image domain. Here we try to do this by minimizing the following loss function instead of Eq. (1):

$$E(\boldsymbol{w}) = \frac{1}{2}||\boldsymbol{w} - \boldsymbol{w}_0||_2^2 + \lambda E_R(\boldsymbol{w}), \tag{6}$$

where the embedding loss $E_R(\boldsymbol{w})$ is minimized while minimizing the difference between the modified parameter $\boldsymbol{w}$ and the original parameter $\boldsymbol{w}_0$. Table 4 summarizes the embedding results after minimizing Eq. (6) against the host network trained on the CIFAR-10 dataset. We can see that embedding fails for $\lambda \leq 1$ as the bit error rate (BER) is larger than zero while the test error of the original task becomes too large for $\lambda > 1$. Thus, it is not effective to directly embed a watermark without considering the original task.

Table 4 Losses, test error (%), and bit error rate (BER) after embedding a watermark with different λ

λ	$\frac{1}{2}\|\|\boldsymbol{w} - \boldsymbol{w}_0\|\|_2^2$	$E_R(\boldsymbol{w})$	Test error	BER
0	0.000	1.066	8.04	0.531
1	0.184	0.609	8.52	0.324
10	1.652	0.171	10.57	0.000
100	7.989	0.029	13.00	0.000

4.3.2 Test error and training loss

Figure 3 shows the training curves for the host network in CIFAR-10 as a function of epochs. Not embedded is the case where the host network is trained without the embedding regularizer. Embedded (direct), Embedded (diff), and Embedded (random), respectively, represent training curves with embedding regularizers whose parameters are $\boldsymbol{X}^{\text{direct}}$, $\boldsymbol{X}^{\text{diff}}$, and $\boldsymbol{X}^{\text{random}}$. We can see that the training loss $E(\boldsymbol{w})$ with a watermark becomes larger than the not-embedded case if the parameters $\boldsymbol{X}^{\text{direct}}$ and $\boldsymbol{X}^{\text{diff}}$ are used. This large training loss is dominated by the embedding loss $E_R(\boldsymbol{w})$, which indicates that it is difficult to embed a watermark directly into a parameter or even into the difference of two parameters. On the other hand, the training loss of Embedded (random) is very close to that of Not embedded.

Table 5 shows the best test errors and embedding losses $E_R(\boldsymbol{w})$ of the host networks with and without embedding. We can see that the test errors of Not embedded and random are

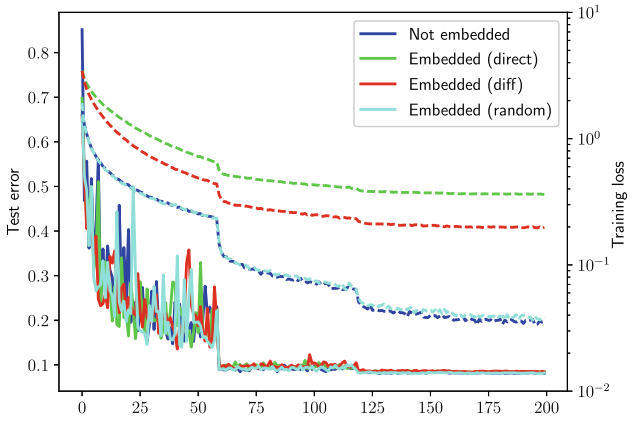

Fig. 3 Training curves for the host network on CIFAR-10 as a function of epochs. Solid lines denote test error (*y*-axis on the left) and dashed lines denote training loss $E(\boldsymbol{w})$ (*y*-axis on the right)

Table 5 Test error (%) and embedding loss $E_R(\boldsymbol{w})$ with and without embedding

	Test error	$E_R(\boldsymbol{w})$
Not embedded	8.04	N/A
Direct	8.21	1.24×10^{-1}
Diff	8.37	6.20×10^{-2}
Random	7.97	4.76×10^{-4}

almost the same, while those of direct and diff are slightly larger. The embedding loss $E_R(\boldsymbol{w})$ of random is extremely low compared with those of direct and diff. These results indicate that the random approach can effectively embed a watermark without impairing the performance in the original task.

4.3.3 Fine-tune-to-embed and distill-to-embed

In the above experiments, a watermark was embedded by training the host network from scratch (train-to-embed). Here, we evaluated the other two situations introduced in Sect. 2.2: fine-tune-to-embed and distill-to-embed.

For fine-tune-to-embed, two experiments were performed. In the first experiment, the host network was trained on the CIFAR-10 dataset without embedding and then fine-tuned on the same CIFAR-10 dataset with and without embedding (for comparison). In the second experiment, the host network is trained on the Caltech-101 dataset and then fine-tuned on the CIFAR-10 dataset with and without embedding.

Table 6a shows the result of the first experiment. Not embedded 1st corresponds to the first training without embedding. Not embedded 2nd corresponds to the second training without embedding and Embedded corresponds to the second training with embedding. Figure 4 shows the training curves of these fine-tunings.[6] We can see that Embedded achieved almost the same test error as Not embedded 2nd and a very low $E_R(\boldsymbol{w})$.

Table 6 Test error (%) and embedding loss $E_R(\boldsymbol{w})$ with and without embedding in fine-tuning and distilling

	Test error	$E_R(\boldsymbol{w})$
(a) Fine-tune-to-embed (CIFAR-10 → CIFAR-10)		
Not embedded 1st	8.04	N/A
Not embedded 2nd	7.66	N/A
Embedded	7.70	4.93×10^{-4}
(b) Fine-tune-to-embed (Caltech-101 → CIFAR-10)		
Not embedded 2nd	7.93	N/A
Embedded	7.94	4.83×10^{-4}
(c) Distill-to-embed (CIFAR-10 → CIFAR-10)		
Not embedded 1st	8.04	N/A
Not embedded 2nd	7.86	N/A
Embedded	7.75	5.01×10^{-4}
(d) Distill-to-embed (CIFAR-10 → Caltech-101)		
Not embedded 1st	8.04	N/A
Embedded	28.34	5.80×10^{-3}

Table 6b shows the results of the second experiment. Not embedded 2nd corresponds to the second training without embedding and Embedded corresponds to the second training with embedding. Figure 5 shows the training curves of these fine-tunings. The test error and training loss of the first training are not shown because they are not compatible with the two different training datasets. From these results, it was also confirmed that Embedded achieved almost the same test error as Not embedded 2nd and very low $E_R(\boldsymbol{w})$. Thus, we can say that the proposed method is effective even in the fine-tune-to-embed situation (in the same and different domains).

Finally, embedding a watermark in the distill-to-embed situation was evaluated. The host network is first trained on the CIFAR-10 dataset without embedding. Then, the trained network was further fine-tuned on the same CIFAR-10 dataset with and without embedding. In this second training, the training labels of the CIFAR-10 dataset were *not* used. Instead, the predicted values of the trained network were used as soft targets [23]. In other words, no label was used in the second training. Table 6c shows the results for the distill-to-embed situation. Not embedded 1st corresponds to the first training and Embedded (Not embedded 2nd) corresponds to the second distilling training with embedding (without embedding). It was found that the proposed method also achieved low test error and $E_R(\boldsymbol{w})$ in the

[6] Note that the learning rate was also initialized to 0.1 at the beginning of the second training, while the learning rate was reduced to (8.0×10^{-4}) at the end of the first training.

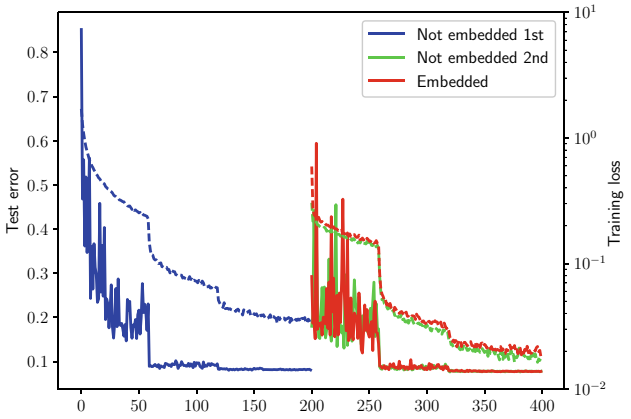

Fig. 4 Training curves for fine-tuning the host network. The first and second halves of epochs correspond to the first and second trainings. Solid lines denote test error (*y*-axis on the left) and dashed lines denote training loss (*y*-axis on the right)

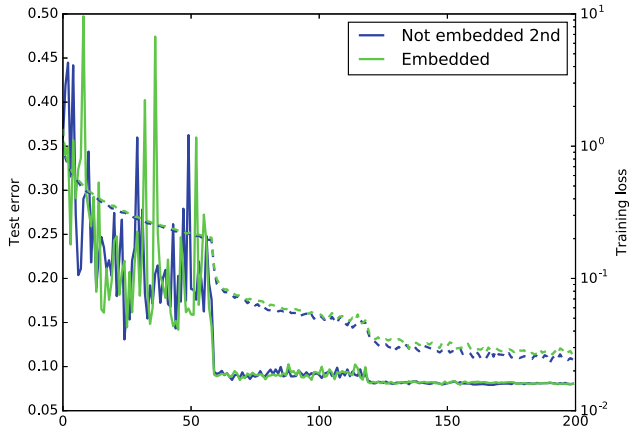

Fig. 5 Training curves for the host network on CIFAR-10 as a function of epochs. Solid lines denote test error (*y*-axis on the left) and dashed lines denote training loss (*y*-axis on the right)

distill-to-embed situation. Table 6d shows the result for the distill-to-embed situation on the different domain; the difference from Table 6c is that the predicted values for the Caltech-101 are used as soft targets here instead of CIFAR-10. The test error is calculated on CIFAR-10.

4.4 Robustness of embedded watermarks

In this section, the robustness of the proposed watermark is evaluated for the three types of attacks explained in Sect. 2.3: fine-tuning, model compression, and watermark overwriting.

4.4.1 Robustness against fine-tuning

Fine-tuning or transfer learning [42] seems to be the most likely type of (unintentional) attack because it is frequently performed on trained models to apply them to other but similar tasks with less effort than training a network from scratch or to avoid over-fitting when sufficient training data are not available.

In this experiment, two trainings were performed; in the first training, a 256-bit watermark was embedded in the conv 2 group in the train-to-embed manner, and then, the host network was further fine-tuned in the second training without embedding, to determine whether or not the watermark embedded in the first training stayed in the host network, even after the second training (fine-tuning).

Table 7 shows the embedding loss before fine-tuning ($E_R(\boldsymbol{w})$) and after fine-tuning ($E'_R(\boldsymbol{w})$), and the best test error after fine-tuning. In the same domain, the host network is trained on the CIFAR-10 dataset while embedding a watermark and then further fine-tuned without embedding a watermark. We evaluated fine-tuning in the same domain (CIFAR-10 → CIFAR-10) and in the different domains (Caltech-101 → CIFAR-10). We can see that, in both cases, the embedding loss was increased slightly by fine-tuning but was still low. In addition, the bit error rate of the detected watermark was equal to zero in both cases. The reason why the embedding loss in fine-tuning in the different domains is higher than that in the same domain is that the Caltech-101 dataset is significantly more difficult than the CIFAR-10 dataset in our settings; all images in the Caltech-101 dataset were resized to 32×32[7] for compatibility with the CIFAR-10 dataset.

4.4.2 Robustness against model compression

It is sometimes difficult to deploy deep neural networks in embedded systems or mobile devices because they are both computationally intensive and memory intensive. In order to solve this problem, the model parameters are often *compressed* [18–20]. The compression of model parameters can intentionally or unintentionally act as an attack against watermarks. In this section, we evaluate the robustness of our watermarks against model compression, in particular, against parameter pruning [20] and distillation [23].

Robustness against parameter pruning In parameter pruning, parameters whose absolute values are very small are cut off to zero. In [19], quantization of weights and the Huffman coding of quantized values are further applied. Because quantization has less impact than parameter pruning and the Huffman coding is lossless compression, we focus on parameter pruning.

In order to evaluate robustness against parameter pruning, we embedded a 256-bit watermark in the conv 2 group while training the host network on the CIFAR-10 dataset. We removed $\alpha\%$ of the $3 \times 3 \times 64 \times 64$ parameters of the embedded layer and calculated embedding loss and bit error

[7] This size is extremely small compared with their original sizes (roughly 300×200).

Table 7 Embedding loss before fine-tuning ($E_R(\boldsymbol{w})$) and after fine-tuning ($E'_R(\boldsymbol{w})$), and the best test error (%) and bit error rate (BER) after fine-tuning

	$E_R(\boldsymbol{w})$	$E'_R(\boldsymbol{w})$	BER	Test error
CIFAR-10 → CIFAR-10	4.76×10^{-4}	8.66×10^{-4}	0.00	7.69
Caltech-101 → CIFAR-10	5.96×10^{-3}	1.56×10^{-2}	0.00	7.88

rate. Figure 6a shows embedding loss $E_R(\boldsymbol{w})$ as a function of pruning rate α. Ascending (Descending) represents embedding loss when the top α% parameters are cut off according to their absolute values in ascending (descending) order. Random represents embedding loss where α% of parameters are randomly removed. Ascending corresponds to parameter pruning, and the others were evaluated for comparison. We can see that the embedding loss of Ascending increases more slowly than those of Descending and Random as α increases. It is reasonable that model parameters with small absolute values have less impact on a detected watermark because the watermark is extracted from the dot product of the model parameter w and the constant embedding parameter (weight) $\boldsymbol{X}$.

Figure 6b shows the bit error rate as a function of pruning rate α. Surprisingly, the bit error rate was still zero after removing 65% of the parameters and 2/256 even after 80% of the parameters were pruned (Ascending). We can say that the embedded watermark is sufficiently robust against parameter pruning because, in [19], the resulting pruning rate of convolutional layers ranged from 16 to 65% for the AlexNet [33], and from 42 to 78% for VGGNet [42]. Furthermore, this degree of bit error can be easily corrected by an error correction code (e.g., the BCH code). Figure 7 shows the histogram of the detected watermark $\sigma(\sum_i X_{ji} w_i)$ after pruning for $\alpha = 0.8$ and 0.95. For $\alpha = 0.95$, the histogram of the detected watermark is also shown for the host network into which no watermark is embedded. We can see that many of $\sigma(\sum_i X_{ji} w_i)$ are still close to one for the embedded case, which might be used as a confidence score in determining the existence of a watermark (zero-bit watermarking).

Robustness against distillation Distillation is a training procedure initially designed to train a deep neural networks model using knowledge transferred from a different model. The intuition was suggested in [4], while distillation itself was formally introduced in [23]. Distillation is employed to reduce computational complexity or compressing the knowledge in an ensemble of models into a single small model. In the standard distillation framework, a large network (or multiple networks) is first trained, and then, a smaller network is trained using the predicted labels of the large network in order to compress the large network. As well as fine-tuning, distillation could be an unintentional attack and it is specific to deep neural networks.

In this experiment, we performed two trainings. First a 256-bit watermark was embedded in the conv 2 group in the

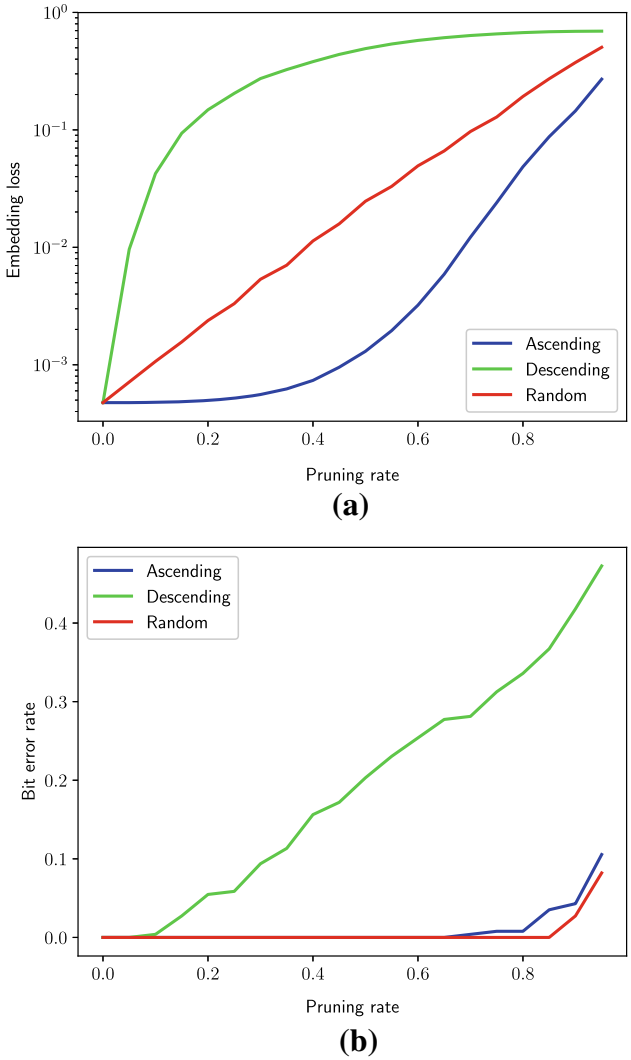

Fig. 6 Embedding loss and bit error rate after pruning as a function of pruning rate. **a** Embedding loss and **b** bit error rate

train-to-embed manner with CIFAR-10. Then, in the second training, another model was distilled using the CIFAR-10 dataset and the predicted values of the first trained network instead of the actual labels. The second training did not embed a watermark and initial weights were set at random. We employed the simplest form of distillation in this experiment. Although we could use a different network architecture and different dataset in the transfer step, we trained a new model of the same architecture on the same set CIFAR-10 for simplicity.

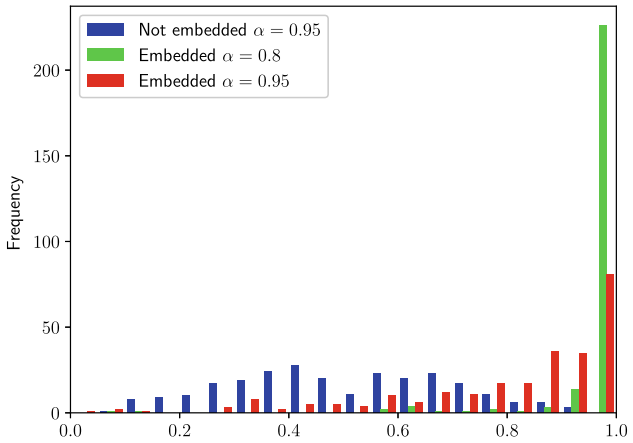

Fig. 7 Histogram of the detected watermark $\sigma(\sum_i X_{ji} w_i)$ after pruning

Table 8 Test error (%) and bit error rate (BER) of the embedded host network and after distilling without embedding the watermark

	Test error	BER
Embedded 1st	8.05	0.00
After distillation	8.40	0.54

Table 8 shows the test error and bit error rate after the first and second trainings. The watermark could not be detected from the distilled model as expected because the model weights had been initialized with random weights.

4.4.3 Robustness against watermark overwriting

Overwriting is a common attack in digital content watermarking [28]. A third-party user may embed a different watermark in order to overwrite the original watermark. Basically, it is necessary to know where the original watermark is embedded to overwrite watermarks. Please note that in addition to regularizer parameters $\boldsymbol{X}$, which work as a secret key, the location where a digital watermark is embedded should also be secret information. However, it is conceivable for a watermark to be embedded into all or multiple layers to destroy the embedded original watermark or change ownership without exact information on where the original watermark is actually embedded.

In order to evaluate robustness against overwriting, we embedded a 256-bit watermark in the conv 2, conv 3 and conv 4 groups with a regularizer parameter $\boldsymbol{X}_0$, while training the host network on the CIFAR-10 dataset. Then, we additionally embedded a 256-bit, 512-bit, 1024-bit, and 2048-bit watermark into the host network, respectively, with a regularizer parameter $\boldsymbol{X}_0$ different from $\boldsymbol{X}_1$. The number of parameters $\boldsymbol{w}$ of conv 2, conv 3, and conv 4 groups were 576, 1152, and 2304, respectively. All bit error rates of the original host networks were zero. The additional watermarks were embedded while training on the CIFAR-10 dataset.

Table 9 Test error (%), embedding loss $E_R(\boldsymbol{w})$, and bit error rate with the original regularizer parameter after overwriting a watermark

Embedded bits	Embedded group		
	conv 2	conv 3	conv 4
(a) Test error (%)			
256	7.43	7.36	7.96
512	7.29	7.35	7.92
1024	7.58	7.41	7.96
2048	7.36	7.61	7.94
(b) Embedding loss			
256	1.67	2.05×10^{-1}	4.98×10^{-2}
512	4.28	1.13	1.94×10^{-1}
1024	1.77×10^{1}	3.76	5.24×10^{-1}
2048	1.04	1.12×10^{1}	1.40
(c) Bit error rate			
256	3.09×10^{-1}	8.59×10^{-2}	3.90×10^{-3}
512	4.10×10^{-1}	2.38×10^{-1}	6.64×10^{-2}
1024	5.11×10^{-1}	4.29×10^{-1}	1.99×10^{-1}
2048	5.27×10^{-1}	5.07×10^{-1}	3.55×10^{-1}

The number of parameters $\boldsymbol{w}$ of conv 2, conv 3, and conv 4 groups is 576, 1152, and 2304, respectively

Table 9 shows test error, embedding loss $E_R(\boldsymbol{w})$, and bit error rate with the first regularizer parameter $\boldsymbol{X}_0$ after overwriting the first watermark. When the bit error rate is close to 0.5, it indicates that the original watermark has been erased completely. We can see that the original watermark was erased in some cases where the number of embedded bits was large compared to the number of parameters $\boldsymbol{w}$.

4.5 Capacity of watermark

In this section, the capacity of the embedded watermark is explored by embedding different sizes of watermarks into different groups in the train-to-embed manner. Please note that the number of parameters $\boldsymbol{w}$ of conv 2, conv 3, and conv 4 groups was 576, 1152, and 2304, respectively. Table 10 shows test error (%), embedding loss $E_R(\boldsymbol{w})$ and bit error rate for combinations of different embedded blocks and different numbers of embedded bits. We can see that embedded loss or test error becomes high if the number of embedded bits becomes larger than the number of parameters $\boldsymbol{w}$ (e.g., 2048 bits in conv 3) because the embedding problem becomes overdetermined in such cases. Thus, the number of embedded bits should be smaller than the number of parameters $\boldsymbol{w}$, which is a limitation of the embedding method using a single-layer perceptron. This limitation would be resolved by using a multilayer perceptron in the embedding regularizer.

Table 10 Test error (%), embedding loss $E_R(\boldsymbol{w})$, and bit error rate for the combinations of embedded groups and sizes of embedded bits

Embedded bits	Embedded group		
	conv 2	conv 3	conv 4
(a) Test error (%)			
256	7.97	7.98	7.92
512	8.47	8.22	7.84
1024	8.43	8.12	7.84
2048	8.17	8.93	7.75
(b) Embedding loss			
256	4.76×10^{-4}	7.20×10^{-4}	1.10×10^{-2}
512	8.11×10^{-4}	8.18×10^{-4}	1.25×10^{-2}
1024	6.74×10^{-2}	1.53×10^{-3}	1.53×10^{-2}
2048	5.35×10^{-1}	3.70×10^{-2}	3.06×10^{-2}
(c) Bit error rate			
256	0.00	0.00	0.00
512	0.00	0.00	0.00
1024	0.00	0.00	0.00
2048	0.28	0.00	0.00

The number of parameters $\boldsymbol{w}$ of conv 2, conv 3, and conv 4 groups was 576, 1152, and 2304, respectively

5 Discussion

5.1 Insights

Fidelity As mentioned in Sect. 3.2, poor local minima are rarely a problem with large networks in practice. Regardless of the initial conditions, the system nearly always reaches solutions of very similar quality. Recent theoretical and empirical results strongly suggest that local minima are not a serious issue in general [35]. Therefore, the proposed approach was able to maintain the performance of the original task and carry out successful watermarking as shown in the experimental results of Sects. 4.3.2 and 4.3.3.

Robustness For watermarking techniques in the neural networks domain, fine-tuning seems to be the most feasible and significant attack. The experimental results in Sect. 4.4.1 show the proposed method could retain the watermark completely after fine-tuning in both cases: the same domain and a different domain. In the case of the same domain, updates of weight values were assumed to be small if the host model was trained well in the first training. On the other hand, in the case of a different domain, weight values are supposed to change dramatically. However, our experimental results show the watermark remained after fine-tuning to a different domain. It is considered that fine-tuning would cause less alteration for weights near the input layer compared to near the output layer. Therefore, the digital watermark could successfully resist a fine-tuning attack, if the watermark is embedded near the input layer of sufficiently deep networks. Additionally, there is an advantage that the network configuration near the input layer may not be changed for another task.

Capacity The result presented in Sect. 4.5 indicates that the capacity is strongly related to the number of the host weights compared to the length of watermarks. Capacity may be increased by using a multilayer perceptron in the embedding regularizer.

5.2 Limitations

Although we have obtained some initial insights into the new problem of embedding a watermark in deep neural networks, the proposed approach still has the following limitations.

Distillation Distillation is theoretically a serious attack for watermarking of neural networks. However, distillation does not seem to be an important attack in reality, since it requires data that are very similar to the inputs used in the original training phase in order to maintain fidelity.

Overwriting As shown in Sect. 4.4.3, overwriting destroys the original watermark. This experiment is assumed to know exactly where the original watermark was embedded. It is conceivable that watermarks could be embedded into all or multiple layers to destroy the original watermark, although this would incur a much greater computational cost due to the large size of widely targeted parameters. Overwriting is still a serve attack, and we should explore an effective way of combatting overwriting.

Black-box type situation In the proposed digital watermarking approach for deep neural network models, we make an assumption that the weight values are visible. Thus, it is impossible to detect abuse in a black-box type situation such as a client–server system where a watermarked model is used on a server by unauthorized parties. To effectively deal with such a situation, the copyright protection of neural network models requires another approach. Inspired by our work [48], Merrer et al. propose a method that allows the extraction of the watermark from a neural network remotely through a service API [38]. The method embeds zero-bit watermarks into models with a stitching algorithm based on adversaries.

5.3 Further expected developments

Further developments are expected by using the analogy of digital content protection and domain-specific issues for deep neural networks.

Embedding as sequential learning In Sect. 4.3.1, we have shown that it is not effective to directly embed a watermark without considering the original task. We can consider this embedding process as sequential learning; the training of the original task is the first task, and subsequent watermark embedding is the second task. Thus, the increase in error rate after embedding can be interpreted as *catastrophic for-*

getting [26]. From this point of view, we can adopt recently developed methods [26,37] to overcome this catastrophic forgetting in embedding watermark.

Compression as embedding Compressing deep neural networks is a very important and active research topic. While we confirmed in this paper that our watermark is very robust against parameter pruning in this paper, a watermark might be embedded in conjunction with compressing models. For example, in [19], after parameter pruning, the network is retrained to learn the final weights for the remaining sparse parameters. Our embedding regularizer can be used in this re-training to embed a watermark.

Network morphism In [9,51], a systematic study has been conducted on how to morph a well-trained neural network into a new one so that its network function can be completely preserved for further training. This network morphism can constitute a severe attack against our watermark because it may be impossible to detect the embedded watermark if the topology of the host network undergoes major modification. We have left the investigation into how the embedded watermark is affected by this network morphism as a topic for future work.

Steganalysis Steganalysis [31,41] is a method for detecting the presence of secretly hidden data (e.g., steganography or watermarks) in digital media files such as images, video, audio, and, in our case, deep neural networks. Watermarks ideally are robust against steganalysis. While, in this paper, we confirmed that embedding watermarks does not significantly change the distribution of model parameters, more exploration is needed to evaluate robustness against steganalysis. Conversely, developing effective steganalysis against watermarks for deep neural networks could be an interesting research topic.

Fingerprinting Digital fingerprinting is an alternative to the watermarking approach for persistent identification of images [6], video [29,47], and audio clips [3,17]. In this paper, we focused on one of these two important approaches. Robust fingerprinting of deep neural networks is another and complementary direction to protect deep neural network models.

6 Conclusions

In this paper, we have proposed a general framework for embedding a watermark in deep neural network models to protect the rights to the trained models. First, we formulated a new problem: embedding watermarks into deep neural networks. We also defined requirements, embedding situations, and the types of attacks that watermarking deep neural networks are vulnerable to. Second, we proposed a general framework for embedding a watermark in model parameters using a parameter regularizer. Our approach does not impair the performance of networks into which a watermark is embedded. Finally, we performed comprehensive experiments to reveal the potential of watermarking deep neural networks as the basis of this new problem. We showed that our framework could embed a watermark without impairing the performance of a deep neural network. The embedded watermark did not disappear even after fine-tuning or parameter pruning; the entire watermark remained even after 65% of the parameters were pruned.

References

1. Abadi M et al (2016) Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv:1603.04467
2. Amari S (1967) A theory of adaptive pattern classifiers. IEEE Trans Electron Comput EC-16 3:299–307
3. Anguera X, Garzon A, Adamek T (2012) Mask: robust local features for audio fingerprinting. In: Proceedings of ICME
4. Ba LJ, Caruana R (2014) Do deep nets really need to be deep? In: Proceedings of NIPS, pp 2654—2662
5. Babenko A, Slesarev A, Chigorin A, Lempitsky V (2014) Neural codes for image retrieval. In: Fleet D, Pajdla T, Schiele B, Tuytelaars T (eds) In: Proceedings of ECCV, pp 584–599
6. Barr J, Bradley B, Hannigan BT (2003) Using digital watermarks with image signatures to mitigate the threat of the copy attack. In: Proceedings of ICASSP, pp 69–72
7. Bengio Y, Courville A, Vincent P (2013) Representation learning: a review and new perspectives. IEEE Trans Pattern Anal Mach Intell 35(8):1798–1828
8. Bergstra J, Breuleux O, Bastien F, Lamblin P, Pascanu R, Desjardins G, Turian J, Warde-Farley D, Bengio Y (2010) Theano: a CPU and GPU math expression compiler. In Proceedings of the Python for scientific computing conference (SciPy)
9. Chen T, Goodfellow I, Shlens J (2016) Net2net: accelerating learning via knowledge transfer. In: Proceedings of ICLR
10. Chollet F (2015) Keras, GitHub repository. https://github.com/fchollet/keras. Accessed 05 Feb 2017
11. Choromanska A, Henaff M, Mathieu M, Arous G, LeCun Y (2015) The loss surfaces of multilayer networks. In: Proceedings of AISTATS
12. Collobert R, Kavukcuoglu K, Farabet C (2011) Torch7: a matlab-like environment for machine learning. In: Proceedings of NIPS workshop on BigLearn
13. Cox I, Miller M, Bloom J, Fridrich J, Kalker T (2008) Digital watermarking and steganography. Morgan Kaufmann Publishers Inc., 2nd edn
14. Dauphin Y, Pascanu R, Gulcehre C, Cho K, Ganguli S, Bengio Y (2014) Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In Proceedings of NIPS
15. Erhan D, Bengio Y, Courville A, Manzagol P-A, Vincent P, Bengio S (2010) Why does unsupervised pre-training help deep learning? J. Mach. Learn. Res. 11:625–660
16. Fei-Fei L, Fergus R, Perona P (2004) Learning generative visual models from few training examples: an incremental Bayesian approach tested on 101 object categories. In: Proceedings of CVPR workshop on generative-model based vision
17. Haitsma J, Kalker T (2002) A highly robust audio fingerprinting system. In: Proceedings of ISMIR, pp 107–115
18. Han S, Liu X, Mao H, Pu J, Pedram A, Horowitz MA, Dally WJ (2016) Eie: efficient inference engine on compressed deep neural network. In: Proceedings of ISCA

19. Han S, Mao H, Dally WJ (2016) Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding. In: Proceedings of ICLR
20. Han S, Pool J, Tran J, Dally WJ (2015) Learning both weights and connections for efficient neural networks. In: Proceedings of NIPS
21. Hartung F, Kutter M (1999) Multimedia watermarking techniques. Proc IEEE 87:1079–1107
22. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of CVPR
23. Hinton G, Vinyals O, Dean J (2014) Distilling the knowledge in a neural network. In: Proceedings of NIPS workshop on deep learning and representation learning
24. Hinton GE, Salakhutdinov RR (2006) Reducing the dimensionality of data with neural networks. Science 313(5786):504–507
25. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
26. James K et al (2017) Overcoming catastrophic forgetting in neural networks. PNAS 114(13):3521–3526
27. Jia Y, Shelhamer E, Donahue J, Karayev S, Long J, Girshick R, Guadarrama S, Darrell T (2014) Caffe: convolutional architecture for fast feature embedding. In: Proceedings of MM
28. Johnson N, Duric Z, Jajodia S (2000) Information hiding: steganography and watermarking–attacks and countermeasures. Springer, Berlin
29. Joly A, Frelicot C, Buisson O (2005) Content-based video copy detection in large databases: a local fingerprints statistical similarity search approach. In: Proceedings of ICIP, pp 505–508
30. Karpathy A, Toderici G, Shetty S, Leung T, Sukthankar R, Fei-Fei L (June 2014) Large-scale video classification with convolutional neural networks. In: Proceedings of ECCV
31. Kodovsky J, Fridrich J, Holub V (2012) Ensemble classifiers for steganalysis of digital media. IEEE Trans Inf Forens Secur 7(2):432–444
32. Krizhevsky A (2009) Learning multiple layers of features from tiny images. Tech Report
33. Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: Proceedings of NIPS
34. Krogh A, Hertz JA (1992) A simple weight decay can improve generalization. In: Proceedings of NIPS
35. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436–444
36. Lecun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86:2278–2324
37. Lee S, Kim J, Jun J, Ha J, Zhang B (2017) Overcoming catastrophic forgetting by incremental mome. In: Proceedings of NIPS
38. Merrer EL, Perez P, Trédan G (2017) Adversarial frontier stitching for remote neural network watermarking. arXiv:1711.01894
39. Nesterov Y (1983) A method of solving a convex programming problem with convergence rate o(1/k2). Sov Math Doklady 27(2):372–376
40. Pang L, Zhu S, Ngo CW (2015) Deep multimodal learning for affective analysis and retrieval. IEEE Trans Multimed 17(11):2008–2020
41. Shaohui L, Hongxun Y, Wen G (2003) Neural network based steganalysis in still images. In: Proceedings of ICME
42. Simonyan K, Zisserman A (2015) Very deep convolutional networks for large-scale image recognition. In: Proceedings of ICLR
43. Sutskever I, Martens J, Dahl G, Hinton G (2013) On the importance of initialization and momentum in deep learning. In: Proceedings of ICML, vol 28, pp III–1139–III–1147
44. Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A (2015) Going deeper with convolutions. In: Proceedings of CVPR
45. Tokui S, Oono K, Hido S, Clayton J (2015) Chainer: a next-generation open source framework for deep learning. In: Proceedings of NIPS workshop on machine learning systems
46. Tomáš M, Martin K, Lukáš B, Jan Č, Sanjeev K (2010) Recurrent neural network based language model. In: Proceedings of INTERSPEECH
47. Uchida Y, Agrawal M, Sakazawa S (2011) Accurate content-based video copy detection with efficient feature indexing. In: Proceedings of ICMR
48. Uchida Y, Nagai Y, Sakazawa S, Satoh S (2017) Embedding watermarks into deep neural networks. In: Proceedings of ICMR
49. van den Oord A, Dieleman S, Schrauwen B (2013) Deep content-based music recommendation. In: Burges CJC, Bottou L, Welling M, Ghahramani Z, Weinberger KQ (eds) In: Proceedings of NIPS, pp 2643–2651. Curran Associates, Inc
50. Wan J, Wang D, Hoi SCH, Wu P, Zhu J, Zhang Y, Li J (2014) Deep learning for content-based image retrieval: a comprehensive study. In: Proceedings of MM, pp 157–166
51. Wei T, Wang C, Rui Y, Chen CW (2016) Network morphism. In: Proceedings of ICML
52. Zagoruyko S, Komodakis N (2016) Wide residual networks. In: Proceedings of ECCV
53. Zhang GP (2003) Time series forecasting using a hybrid arima and neural network model. Neurocomputing 50:159–175